RADC-TR-83-83
In-House Report
March 1983

PROBABILITY EXPRESSIONS, WITH APPLICATIONS TO FAULT TESTING IN DIGITAL NETWORKS

Warren H. Debany, Jr.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-83-83 has been reviewed and is approved for publication.

APPROVED: Joseph B. Brauer

JOSEPH B. BRAUER, Chief
Microelectronics Reliability Branch
Reliability & Compatibility Division

APPROVED: Anthony J. Feduccia

ANTHONY J. FEDUCCIA, Acting Chief
Reliability & Compatibility Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBRA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-83-83	2. GOVT ACCESSION NO. AD-A131367	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROBABILITY EXPRESSIONS, WITH APPLICATIONS TO FAULT TESTING IN DIGITAL NETWORKS		5. TYPE OF REPORT & PERIOD COVERED In-House Report 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Warren H. Debany, Jr.		8. CONTRACT OR GRANT NUMBER(s) N/A
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rome Air Development Center (RBRA) Griffiss AFB NY 13441		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 2338013P
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (RBRA) Griffiss AFB NY 13441		12. REPORT DATE March 1983 13. NUMBER OF PAGES 102
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Boolean Algebra	NP-Complete	Software
Combinational Network	Path Sensitization	Test Generation
Computer Programs	LASAR	
Fault Detection	Probability	
Modeling	Random Testing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Logical functions are usually described in terms of Boolean Expressions. In this report we employ an alternate representation for logical functions, namely in terms of Probability Expressions (P-exps). P-exps are algebraic expressions that yield the probability that an output signal takes the logical value 1, given the independent probabilities that the input signals take the value 1. In a manner similar to signal representation in time and frequency domains, we may represent logical functions in two domains: (over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Cont'd

Boolean and Probabilistic domains.

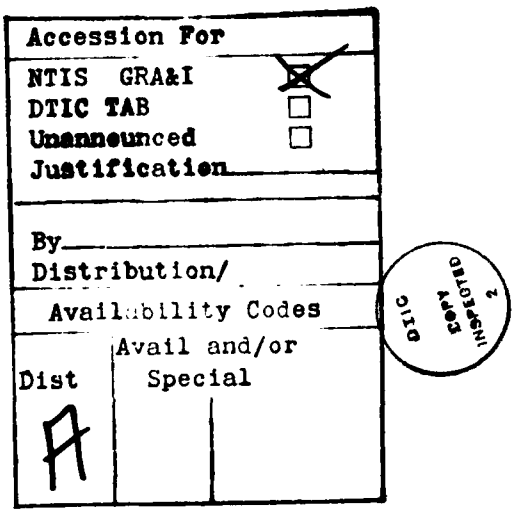
In this report we review and present some properties of P-exps. Computational aspects of P-exps are discussed extensively, including difficulty of computer manipulation, lengths of P-exps, solution of P-exps, and transformations between the two domains. P-exps provide a computational savings in certain classes of problems. P-exps are then applied to the problem of test generation for combinational networks. Further, we derive some results concerning the random testing of combinational networks.

ACKNOWLEDGEMENTS

The author wishes to thank Sue Desens for the help she gave in the formatting and preparation of this report. Allen Converse wrote the software that aided in the formatting of this report.

Professor Carlos Hartmann deserves thanks for the direction he gave in the algorithmic complexity and testing discussions. Professor Donald Weiner and Professor Harry Schwartzlander, serving as members of the author's thesis committee, made valuable comments and corrected several errors.

It would be impossible to describe the contributions made by Professor Pramod K. Varshney. Serving as the author's thesis advisor he provided direction, advice, and extraordinary technical review of the investigations described in this report. His contributions of time and effort went far beyond what was required by his job. The author's most sincere thanks go to him.



SUMMARY

Logical functions are usually described in terms of Boolean Expressions. In this report we employ an alternate representation for logical functions, namely in terms of Probability Expressions (P-exps). P-exps are algebraic expressions that yield the probability that an output signal takes the logical value 1, given the independent probabilities that the input signals take the value 1. In a manner similar to signal representation in time and frequency domains, we may represent logical functions in two domains: Boolean and Probabilistic domains.

In this report we review and present some properties of P-exps. Computational aspects of P-exps are discussed extensively, including difficulty of computer manipulation, lengths of P-exps, solution of P-exps, and transformations between the two domains. P-exps provide a computational savings in certain classes of problems. P-exps are then applied to the problem of test generation for combinational networks. Further, we derive some results concerning the random testing of combinational networks.

This report was submitted to the Electrical and Computer Engineering Department of Syracuse University, Syracuse NY, as a Master's Thesis. It is published here in a slightly different form. The software developed as part of this work is owned by the Air Force.

TABLE OF CONTENTS

I. INTRODUCTION

1.1 BOOLEAN EXPRESSIONS AND PROBABILITY EXPRESSIONS

A logical function describes the logical relationship between a set of inputs and an output. The output of a logical function is completely determined by the values taken by the input variables. Generally, input variables are considered to take values independently of each other. For logical functions defined over GF(2) the input and output alphabet consists of the elements 0 and 1. A logical function can be described in many forms, such as a truth table, a list of minterms or maxterms, or in the algebraic form. The algebraic representation of logical functions is in terms of Boolean expressions (B-exps). A single logical function may be represented by many equivalent B-exps. All of these equivalent B-exps, however, can be reduced to unique standard or canonical forms. A Sum-Of-Products (SOP) form containing only minterms is known as the Disjunctive Normal Form (DNF), and a Product-Of-Sums (POS) form containing only maxterms is known as the Conjunctive Normal Form (CNF). Both the DNF and CNF are unique for every logical function (Kohavi [37]).

For any logical network, composed of AND, OR, NOT, NAND, NOR, EXOR, or EXNOR gates, one may write a B-exp that represents the state of the output or any internal signal line in the network as a function of the primary inputs to the network. B-exps can be manipulated in a variety of ways. For example, given an input condition (or an "input vector") the logical value of any signal line (or intermediate point) can be determined. Conversely, given a desired internal condition, the B-exps can be solved to find a set of necessary input conditions. The latter problem, in particular, is encountered when generating fault detection tests for logical networks, although it should be mentioned that B-exps are not generally used for test generation since more efficient approaches are available.

B-exps and their manipulation have been treated extensively in the literature (e.g., Kohavi [37]). In this case, the input, output, and internal variables take on the value 0 or 1 and the manipulations are performed in accordance with the rules of Boolean algebra (see Table 1). There is some recent interest in the probabilistic treatment of logical functions [1-5,46-49,54-55]. The inputs to a logical network (or the independent variables of a B-exp) are specified as an independent vector of probabilities with which the corresponding input variable takes the logical value 1. In a similar manner, we can associate a probability measure with the internal and the output variables. We will use the term signal probability to refer to the probability that the signal in question takes the logical value 1. The signal

probabilities of the output and the internal variables can be related to the signal probabilities of the input variables. This relationship can be expressed in terms of Probability expressions (P-exps). These P-exps can be manipulated using the rules of Probability Theory. As we will observe, P-exps are related to B-exps and, in fact, can be derived from B-exps. Thus, we have another representation for logical functions that can be considered as a representation in another "domain." B-exps are defined in the Boolean domain (B-domain) and P-exps are defined in the Probabilistic domain (P-domain). In the B-domain, B-exps obey rules of Boolean algebra whereas in the P-domain, P-exps follow the rules of Probability Theory. The values taken by the logical function in the B-domain are 0 or 1 whereas in the P-domain a logical function can take any real value over the closed interval [0.0, 1.0] (real values will always be written with a decimal point, to avoid confusion with logical values). We should note the analogy of the representation of logical functions in two domains with the time and frequency domain representations of signals.

The goal in this report is to present and apply the theory of logical function representation in the P-domain. Relationships between B-exps and P-exps will be examined, and we will define the transformations used to convert expressions from one domain to the other. We will utilize the uniqueness of these transforms. The theorems dealing with the properties of P-exps will be used in the manipulation of P-exps. Computational complexity of this manipulation will be of concern. It is expected that the representation of logical functions in terms of P-exps and their manipulation in the P-domain would be helpful in several applications. One such area, which is treated in this report, is test generation and testability for logical networks. Specifically, we can answer questions such as fault latency, difficulty of path sensitization, length of test sets, or determination of favorable conditions for random testing.

1.2 LITERATURE SUMMARY

As early as 1854, G. Boole, in the Laws of Thought [14], used a combination of logical analysis and the laws of probability to solve probability-related word problems. Unfortunately, the text is cryptic and there are many leaps of intuition. This work was later clarified by Hailperin [28], which will be discussed later in this section.

In 1955, C. Y. Lee [38] discussed the application of probability theory to the determination of reliability of a telephone switching network. His procedure for producing "generating functions for linking probabilities" uses the properties of P-exps. These functions described the probability that a complete path existed between two points in a graph, given

the probabilities of path failure. He explicitly showed the use of idempotency in taking probabilistic products. Lee pointed out that these expressions for linking probabilities are difficult to expand. We will show later that this is at least an NP-complete problem.

In 1963, R. B. Hurley [32] used Boolean algebra and the laws of probability to solve problems in reliability. He considered blocks (which represented systems) in series, parallel, series-parallel, M-out-of-N, and bridge configurations, and by using Karnaugh Maps produced probabilistic expressions for reliability.

In 1967, S. T. Ribeiro [49] discussed a "coding scheme" for describing the probability of the occurrence of a pulse in a "random-pulse machine." A random-pulse machine is one in which Boolean values are expressed as pulse probabilities. This was based on the work of von Neumann. Ribeiro gave a table which listed all B-exps and P-exps of two variables, and showed equivalence between the two forms.

A chapter of a book by G. J. Klir [35] is devoted to the analysis and synthesis of probabilistic switching circuits. Although Klir covered sequential as well as purely memoryless circuits, the latter problem is most closely related to the subject of P-exps. The concept of his "Probability Transformer" will be dealt with in a later section in this report, under "Random Function Generators." His Table 10.1 lists P-exps for some functions of one, two, and three variables.

Work by K. P. Parker and E. J. McCluskey [46-47] presented the foundations of P-exps as they are used in this report. Their lemmas provided the first formal description of the properties of P-exps. The authors presented an algorithm for producing P-exps from the DNF of a logical function. They stated (without proof) that there is a unique P-exp for each logical function. They discussed briefly a number of topics such as fault detection, signal reliability, and threshold functions. They presented these topics, however, with "bundled" inputs, by which they meant that all input signals have equal probabilities (not necessarily 1/2). There is a resulting loss of generality, as will be shown later. Parker and McCluskey later observed [48] that P-exps produce the same result as B-exps when the input probabilities are identically 0 and 1. In addition, they presented a second algorithm for generating P-exps from the logical network description, which produces a P-exp for the output of a gate directly from the P-exps for the gate's inputs, rather than resort to using the DNF of the network's function.

A text by T. Hailperin [28] is devoted to analyzing and regularizing the work of Boole [14]. In his introduction Hailperin stated, "The elimination by later logicians of what was not fully understandable or not strictly relevant to the logic of class terms resulted in a calculus whose abstract formulation is now known as the theory of Boolean algebras or, in an equivalent form, the theory of Boolean rings." In the portion of Hailperin's work that is applicable here, he attempts to make a meaningful interpretation of Boole's "system." He coined the term Signed Heap to describe an algebra that is related to the P-exps described here.

A short paper by W. G. Schneeweiss [52] presented the essentials of P-exps from the standpoint of calculating reliability. Remarkably, the comments of two of the referees were published with the paper. The comments by Bennetts [11] were rather strong, criticizing both the rigor of Schneeweiss' notation, and the effectiveness of the P-exp approach. The comments by Bennetts included a network originally presented in [10]. Bennetts generated the P-exp for this network, and although the P-exp was not listed, he stated that it contained 315 terms, which we have verified. He contrasted this with his Reverse-Polish approach in [10] which results in a disjoint B-exp (in SOP) consisting of only 8 terms. Although Bennetts [10] indicated that a computer program was used to generate the 8 disjoint terms no indication of the type of computer, language, or run time was given. It should be mentioned that in running the example by Bennetts, our computer program for performing the P-exp manipulations required 3.15039 CPU seconds to convert a NAND/AND version of the network into a P-exp. Of this time, only 1.70676 CPU seconds were used in the pure P-exp portion of the algorithm (that is to say, excluding parsing of the input format and formatting of output). Our program is written in PL/I, running under the Multics Operating System on a Honeywell 6180. This network will be discussed again in Section 2.4, Example 6.

Kumar and Breuer [36] in 1981 presented a number of the properties of P-exps, based largely on the work of Parker and McCluskey [48]. They further presented a proof of the uniqueness of the P-exp form of a logical function. Kumar and Breuer showed a matrix method of performing the P-Transform. They expressed P-exps in a vector form, derivable from the integer coefficients of the terms in the P-exp, which they called the spectrum, S. They showed how the Walsh coefficients for the logical function [20-21] could be determined directly from S.

1.3 ORGANIZATION OF REPORT

Chapter II of this report presents the basis of the theory of P-exps. Chapter III covers the manipulation of P-exps and introduces techniques for performing an "Inverse P-transform." In order to accomplish the work presented in this report a computer program was written that performs the operations described. Although the details of the software are not given here, Chapter III outlines the data formats, or representations, that were used, and discusses factors relating to algorithmic complexity. Chapter IV is devoted to the application of P-exps to problems in test generation for digital networks. Chapter V is the summary and discussion of the work. The Appendix summarizes the functions of several computer programs that were used in the work described in this report.

1.4 GENERAL NOTATION

Chapter II will describe the specific notation to be used for expressing B-exps and P-exps. It is worth noting at this point a few general items.

The graphic symbols used to denote logic gates are taken from IEEE Std 91-1973, "IEEE Standard Graphic Symbols for Logic Diagrams (Two-State Devices)," also known as ANSI Y32.14-1973. This standard supersedes MIL-STD-806B.

Starting in Chapter III we employ "Order Notation." Rather than define this notation formally, let us give some examples. Suppose that we classify sorting algorithms according to the number of comparisons required. For N data items, the Bubble Sort requires $0.5N^2-0.5N$ comparisons. This would be considered to be an $O(N^2)$ algorithm (read this as "Order N squared" or "On the order of N squared"). To obtain this, we take the term that is largest as N approaches infinity, and discard coefficients. As another example, an algorithm requiring $50N\log_{10}N$ operations would be denoted by O(NlogN), where the base of the logarithm is immaterial.

We will talk about problems that take "polynomial time" for solution, and about others that are "NP-complete" and "NP-hard." The reader who is already familiar with these terms should <u>not</u> read the following explanation, as it is simplistic and largely incorrect. For those readers only who are completely unfamiliar with these terms:

Polynomial Time: Problems with solutions that have complexity of O(N), $O(N^2)$, $O(N^3)$, etc.

NP-Complete and NP-Hard: Problems with solutions that take $O(2^N)$ or O(N!) time, or similar such exponential growth.

II. DERIVATION AND PROPERTIES OF P-EXPS

In this chapter we will present the theory of P-exps. We will draw on the theorems presented in the literature that deal with the properties of P-exps.

2.1 NOTATION

We first define the notation to be used, and point out the meanings of the operations in the two domains.

Lower-case symbols - Boolean variables and expressions, and events.

Upper-case symbols - Probabilistic transforms of the Boolean variables and expressions, and numeric probabilities.

$\overline{a}$ Boolean complement of "a."

$\overline{A}$ 1-A, the additive inverse or P-exp complement function.

\+ OR (for B-exp), or ordinary addition (for P-exp).

(juxtaposition)
AND.

\- Ordinary subtraction.

<--> Transform pair: B-exp <--> P-exp.

P(a,b) Probability of occurrence of events a and b.

P(a v b) Probability of occurrence of event a or b (inclusive OR).

Several of the above operators have different meanings in the B- and P-domains. There will be no ambiguity, due to the use of lower- and upper-case letters to distinguish between the domains. The AND operation for P-exps is closely related to polynomial multiplication. Further discussion will be deferred until Section 2.5.

2.2 LAWS OF PROBABILITY

Consider the probability space

(S, B, P)

where S is a sample space, B is the set of events on S, and P is the probability measure. Let a and b be two events whose probabilities are given by P(a) and P(b). The probability of the complementary event, a, is

$$P(\bar{a}) = 1 - P(a) \tag{1}$$

The probability of the union of the two events is given by

$$P(a \vee b) = P(a) + P(b) - P(a,b) \tag{2}$$

where P(a,b) is the probability of the intersection of the two events. Note that the probability of the union of more than two events can be found by repeated application of Eq. 2, or by the direct application of the General Additive Law of Probability [43], which is also commonly referred to as the "inclusion-exclusion" principle:

$$\begin{aligned} P(a_1 \vee a_2 \vee \ldots \vee a_n) &= \sum_{i=1}^{n} P(a_i) \\ &- \sum_{i<j} \sum P(a_i, a_j) \\ &+ \sum_{i<j<k} \sum \sum P(a_i, a_j, a_k) \\ &- \ldots + \ldots - \ldots + \ldots \\ &-(-1)^n P(a_1, a_2, \ldots a_n) \end{aligned} \tag{3}$$

If events a and b are <u>independent</u>, then the probability of the joint event is simply

$$P(a,b) = P(a)P(b) \tag{4}$$

where the individual probabilities have been multiplied using ordinary multiplication. When events a and b are dependent, the problem is no longer straightforward. In the special case when a = b (i.e., a single event),

$$P(a,b) = P(a,a) = P(a) \tag{5}$$

This important observation will lead to the Idempotency Reduction Rule.

2.3 LAWS OF BOOLEAN ALGEBRA

Some basic laws of Boolean algebra are listed in Table 1. These are found in any text on Switching Theory (such as [26] or [37]). We will use these in the derivations of the P-exps, and since it will be shown that P-exps are unique, properties such as Absorption and Consensus reduce to identities. DeMorgan's Theorem is of particular importance to our later work in the application of P-exps to the testing problem. We will use this theorem implicitly when we convert arbitrary combinational networks to NAND/AND representations. The Shannon Expansion Theorem is used in Chapter III in the development of the Inverse P-transform.

2.4 DEMONSTRATION OF THE P-EXP CONCEPT

Let us now introduce the concept of the P-exp by a simple example:

Example 1: Figure 1 shows a small digital network composed of NAND gates. Consider the gate with output f. For this gate,

$$f = \overline{ab}$$

and if we are given the values of a and b we can calculate the value of f. On the other hand, if we are given probabilities that a and b take the logical probability 1, and are asked to compute the probability that f is a 1, we can either apply techniques such as Hurley's [32], or we can use the P-exp corresponding to the B-exp given above.

Although we have not yet listed the necessary theorems, we will show how the P-exp for f can be generated and used. There is a transform pair

Identity Elements	$a+1 = 1$	$a \cdot 0 = 0$
Inverse Elements	$a+\bar{a} = 1$	$a \cdot \bar{a} = 0$
Commutativity	$a+b = b+a$	$ab = ba$
Distributivity	$a+bc = (a+b)(a+c)$	$a(b+c) = ab+ac$
Associativity	$a+(b+c) = (a+b)+c$	$a(bc) = (ab)c$
Idempotency	$a+a = a$	$aa = a$
Involution	$\bar{\bar{a}} = a$	
Absorption	$a+ab = a$	$a(a+b) = a$
	$a+\bar{a}b = a+b$	$a(\bar{a}+b) = ab$
Consensus	$ab+\bar{a}c+bc = ab+\bar{a}c$	$(a+b)(\bar{a}+c)(b+c)$ $= (a+b)(\bar{a}+c)$
DeMorgan's Thm	$\overline{a+b} = \bar{a}\bar{b}$	$\overline{ab} = \bar{a}+\bar{b}$
Shannon Expansion Thm	$f(a, b, c, \ldots) = a \cdot f(1, b, c, \ldots)$ $+ \bar{a} \cdot f(0, b, c, \ldots)$	
	$= (a + f(0, b, c, \ldots))$ $(\bar{a} + f(1, b, c, \ldots))$	

Table 1. Some Theorems of Boolean Algebra (with duals).

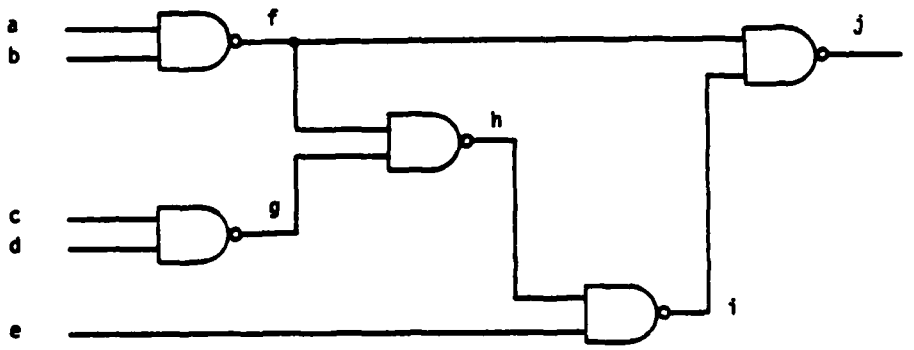


Figure 1. Digital Network for Example 1.

$f \longleftrightarrow F$

such that

$f = \overline{ab} \longleftrightarrow F = \overline{AB} = 1\text{-}AB$

In order to use this P-exp, $F = 1\text{-}AB$, we assign (arbitrarily) the probabilities $A = B = 1/2$. We can then calculate that

$$F = 1 - (1/2)^2$$
$$= 3/4$$

Comparing this to a truth table for a NAND gate shows this to be the correct result.

(End of example)

2.5 TRANSFORMATION THEOREMS

As in [36], P-exps are defined as follows:

For a logical function, f, the P-exp, F, where F = Prob{f = 1}, yields the probability that f equals 1 as a function of the probabilities that its inputs equal 1.

The transformation theorems that follow provide the means by which we can derive F from f. While in principle an inverse-transformation would be possible by the same theorems, this would be impractical. This will be discussed in Section 3.6.

Since these theorems were shown or proved primarily by Parker and McCluskey [46-48] and by Kumar and Breuer [36] we will only state the theorems, omitting the proofs.

Theorem 1 ([46, Lemma 1]): Let a be a Boolean variable or expression, and A be its transform in the P-domain, i.e.,

$a \longleftrightarrow A$

then,

$\overline{a} \leftrightarrow \overline{A} = 1-A$

Theorem 2 (Idempotency Reduction Rule, IRR) ([38, Eq. 2.23]): For a transform pair

$a \leftrightarrow A$

we have,

$aa = a \leftrightarrow AA = A$

Theorem 3 ([46, Lemma 2]): For B-exps a and b, and their corresponding P-exps A and B, we have

$ab \leftrightarrow AB$

where juxtaposition of the P-exps indicates the AND operation.

ANDing of two P-exps is closely related to multiplication. From Eq. 4 two P-exps consisting of the product of independent variables may simply be multiplied together, subject to the IRR (e.g., (AB)(BC) = ABC). The ANDing of a P-exp consisting of a single product term with a P-exp consisting of more than one term is the expression resulting from multiplying the single term by each of the other terms, subject to the IRR and the combining or removing of like terms (e.g., (A)(1-B) = A-AB, or (AB)(1-B) = 0). The ANDing of two P-exps consisting of multiple terms is similar to polynomial multiplication, where every term in the first expression is multiplied by every term in the second, subject to the above conditions (e.g., (1-A)(1-B) = 1-A-B+AB). The final P-exps will always consist only of independent input variables.

Theorem 4 (Inclusion-Exclusion) ([46, Lemma 3]): For B-exps, a and b, and their corresponding P-exps, A and B, we have

$a+b \leftrightarrow A+B-AB$

By repeated application of this theorem, or by direct application of DeMorgan's Theorem the extended forms of this theorem can be obtained, such as:

$a+b+c \leftrightarrow A+B+C-AB-AC-BC+ABC$

The product terms and the signs in the general case are the same as those indicated in Eq. 3, the General Additive Law of Probability.

Example 2: The theorems presented above can be used to convert arbitrary B-exps into P-exps. Consider the logical function $f = \bar{a}b+bc$. We may apply Theorems 2 and 4, to obtain

$$F = \bar{A}B + BC - \bar{A}BC$$

and then expand using Theorem 1,

$$= (1-A)B + BC - (1-A)BC$$
$$= B - AB + BC - BC + ABC$$
$$= ABC + B - AB$$

We can then state that

$$\bar{a}b + bc \leftrightarrow ABC + B - AB$$

(End of example)

Example 3: We show that the Boolean Absorption Theorem (see Table 1) reduces to an identity in the P-domain. From Table 1:

$$a + ab = a$$

We take the P-transform of the left-hand side,

$$f = a + ab \leftrightarrow F = A + AB - AB$$
$$= a \qquad\qquad = A$$

Thus, the results from simplification of the B-exp and of transformation to the P-domain are consistent.

(End of example)

Example 4: We may perform a slightly more complicated example using the Consensus Theorem. Let $f = ab + \bar{a}c + bc$, and $g = ab + \bar{a}c$. From Table 1, f = g. Applying the theorems above,

$$F = AB + \overline{A}C + BC - ABC - \overline{A}BC$$
$$= AB + C - AC + BC - ABC - BC + ABC$$
$$= AB + C - AC$$

similarly,

$$G = AB + \overline{A}C$$
$$= AB + C - AC$$

thus, we see that F and G result in the identical expressions.

(End of example)

Example 5: We can use P-exps to determine the signal values in logical networks. Consider the network shown in Figure 1. Signal j will be a function of the independent variables a, b, c, d, and e. Applying the Boolean rules listed in Table 1:

$$f = \overline{ab}$$

$$g = \overline{cd}$$

$$h = \overline{fg}$$
$$= \overline{f} + \overline{g}$$
$$= ab + cd$$

$$i = \overline{he}$$
$$= \overline{h} + \overline{e}$$
$$= \overline{\overline{ab + cd}} + \overline{e}$$
$$= \overline{ab}\,\overline{cd} + \overline{e}$$
$$= (a+b)(c+d) + \overline{e}$$
$$= \overline{ac} + \overline{ad} + \overline{bc} + \overline{bd} + \overline{e}$$

$$j = \overline{fi}$$
$$= \overline{f} + \overline{i}$$
$$= ab + (a+c)(a+d)(b+c)(b+d)e$$
$$= ab + (a+cd)(b+cd)e$$
$$= ab + abe + acde + bcde + cde$$

= ab + cde

If we perform these operations in the P-domain, we obtain:

F = 1 - AB

G = 1 - CD

H = 1 - FG
= 1 - (1 - AB - CD + ABCD)
= AB + CD - ABCD

I = 1 - EH
= 1 - ABE - CDE + ABCDE

J = 1 - FI
= 1 - (1 - AB)(1 - ABE - CDE + ABCDE)
= 1 - (1 - ABE - CDE + ABCDE
- AB + ABE + ABCDE - ABCDE)
= AB + CDE - ABCDE

It is verified by using Theorem 4 that

ab + cde <--> AB + CDE - ABCDE

(End of example)

Example 5 points out a number of interesting points. First, use of the Boolean identities as reduction rules is difficult even in such a simple case. While expanding the expression for j we noticed that (a+c)(a+d) = a+cd, etc. Otherwise the final number of products would have greatly increased. In the generation of the P-exps, cancellation of a number of terms occurred, and since cancellation (or combination) occurs only with pairs of identical terms, reduction is more straightforward.

Example 6: A more complicated network is shown in Figure 2a. This is the example given by Bennetts [10] that was mentioned in his comments [11] on the paper by Schneeweiss [52]. The resulting 315 term expression is shown in Figure 2b, and obtaining it required less than two

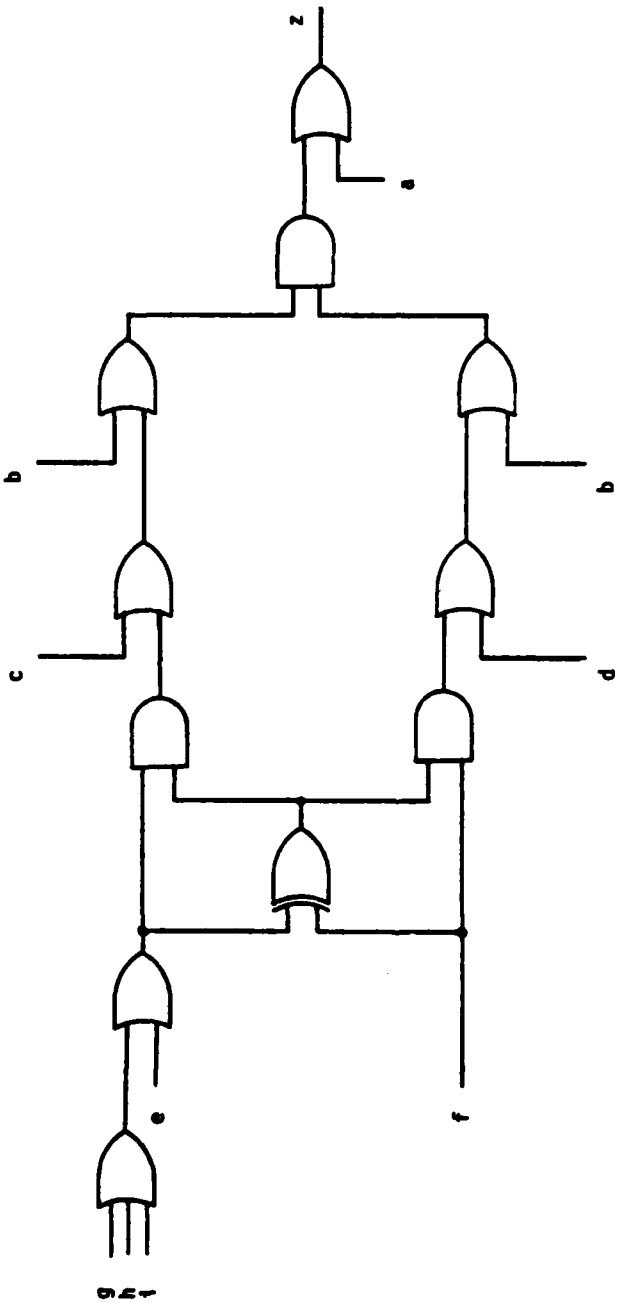


Figure 2a. Fault Tree.

```
a + a b + a b c d + a b c d e f + a b c d e f g + a b c
d e f g h + a b c d e f g h i + a b c d e f g h i

                        <-->

2ABCDEFGH + 2ABCDEFGI + 2ABCDEFHI + 2ABCDEF + ABCDEGHI
+ ABCDEG + ABCDEH + ABCDEI + 2ABCDFGHI + 2ABCDFG
+ 2ABCDFH + 2ABCDFI + ABCDGH + ABCDGI + ABCDHI + ABCD
+ ABCEFGHI + ABCEFG + ABCEFH + ABCEFI + ABCFGH + ABCFGI
+ ABCFHI + ABCF + ABDEFGHI + ABDEFG + ABDEFH + ABDEFI
+ ABDEGH + ABDEGI + ABDEHI + ABDE + ABDFGH + ABDFGI
+ ABDFHI + ABDGHI + ABDG + ABDH + ABDI + 2ACDEFGHI
+ 2ACDEFG + 2ACDEFH + 2ACDEFI + ACDEGH + ACDEGI
+ ACDEHI + ACDE + 2ACDFGH + 2ACDFGI + 2ACDFHI + ACDF
+ ACDGHI + ACDG + ACDH + ACDI + ACEFGH + ACEFGI
+ ACEFHI + ACEF + ACFGHI + ACFG + ACFH + ACFI + ADEFGH
+ ADEFGI + ADEFHI + ADEF + ADEGHI + ADEG + ADEH + ADEI
+ ADFGHI + ADFG + ADFH + ADFI + ADGH + ADGI + ADHI + A
+ 2BCDEFGHI + 2BCDEFG + 2BCDEFH + 2BCDEFI + BCDEGH
+ BCDEGI + BCDEHI + BCDE + 2BCDFGH + 2BCDFGI + 2BCDFHI
+ BCDF + BCDGHI + BCDG + BCDH + BCDI + BCEFGH + BCEFGI
+ BCEFHI + BCEF + BCFGHI + BCFG + BCFH + BCFI + BDEFGH
+ BDEFGI + BDEFHI + BDEF + BDEGHI + BDEG + BDEH + BDEI
+ BDFGHI + BDFG + BDFH + BDFI + BDGH + BDGI + BDHI + B
+ 2CDEFGH + 2CDEFGI + 2CDEFHI + 2CDEF + CDEGHI + CDEG
+ CDEH + CDEI + 2CDFGHI + 2CDFG + 2CDFH + 2CDFI + CDGH
+ CDGI + CDHI + CD + CEFGHI + CEFG + CEFH + CEFI + CFGH
+ CFGI + CFHI + CF + DEFGHI + DEFG + DEFH + DEFI + DEGH
+ DEGI + DEHI + DE + DFGH + DFGI + DFHI + DGHI + DG
+ DH + DI - 2ABCDEFGHI - 2ABCDEFG - 2ABCDEFH - 2ABCDEFI
- ABCDEGH - ABCDEGI - ABCDEHI - ABCDE - 2ABCDFGH
- 2ABCDFGI - 2ABCDFHI - ABCDF - ABCDGHI - ABCDG - ABCDH
- ABCDI - ABCEFGH - ABCEFGI - ABCEFHI - ABCEF - ABCFGHI
- ABCFG - ABCFH - ABCFI - ABDEFGH - ABDEFGI - ABDEFHI
- ABDEF - ABDEGHI - ABDEG - ABDEH - ABDEI - ABDFGHI
- ABDFG - ABDFH - ABDFI - ABDGH - ABDGI - ABDHI - AB
- 2ACDEFGH - 2ACDEFGI - 2ACDEFHI - 2ACDEF - ACDEGHI
- ACDEG - ACDEH - ACDEI - 2ACDFGHI - 2ACDFG - 2ACDFH
- 2ACDFI - ACDGH - ACDGI - ACDHI - ACD - ACEFGHI
- ACEFG - ACEFH - ACEFI - ACFGH - ACFGI - ACFHI - ACF
- ADEFGHI - ADEFG - ADEFH - ADEFI - ADEGH - ADEGI
- ADEHI - ADE - ADFGH - ADFGI - ADFHI - ADGHI - ADG
- ADH - ADI - 2BCDEFGH - 2BCDEFGI - 2BCDEFHI - 2BCDEF
- BCDEGHI - BCDEG - BCDEH - BCDEI - 2BCDFGHI - 2BCDFG
- 2BCDFH - 2BCDFI - BCDGH - BCDGI - BCDHI - BCD
- BCEFGHI - BCEFG - BCEFH - BCEFI - BCFGH - BCFGI
- BCFHI - BCF - BDEFGHI - BDEFG - BDEFH - BDEFI - BDEGH
- BDEGI - BDEHI - BDE - BDFGH - BDFGI - BDFHI - BDGHI
- BDG - BDH - BDI - 2CDEFGHI - 2CDEFG - 2CDEFH - 2CDEFI
- CDEGH - CDEGI - CDEHI - CDE - 2CDFGH - 2CDFGI
- 2CDFHI - CDF - CDGHI - CDG - CDH - CDI - CEFGH
- CEFGI - CEFHI - CEF - CFGHI - CFG - CFH - CFI - DEFGH
- DEFGI - DEFHI - DEF - DEGHI - DEG - DEH - DEI - DFGHI
- DFG - DFH - DFI - DGH - DGI - DHI
```

Figure 2b. 315-term P-exp for Output of Fault Tree.

CPU seconds, as noted in Chapter I. Only the final P-exp is shown, although it was obtained by successive applications of Theorems 1 and 2 to previously-generated P-exps, just as we did in Example 5.

(End of example)

Table 2 lists all $2^{2^2} = 2^4 = 16$ possible functions of two variables, in both B-exp and P-exp form. Note that each of these P-exps is unique. In addition, each P-exp is evaluated with equiprobable inputs, and the fractions in that column correspond exactly to the fraction of the number of minterms equaling 1 for each function. The fact that each of these P-exps is unique suggests that there is no loss of information in the transformation to the P-domain, and that the transform thus has an inverse.

Theorem 5 (Uniqueness) ([36, Theorem 2]): For every B-exp there is a unique P-exp.

From uniqueness, it follows that, as in the B-domain, the AND and OR operations are commutative, associative, and are distributive over each other.

A theorem which will aid us in manipulating P-exps is the following:

Theorem 6 ([36, Theorem 3]): If the probabilities of the input variables to a P-exp are set to exactly 0 and 1, then

$$f(x_1, x_2, x_3, \ldots) = F(X_1, X_2, X_3, \ldots)$$

when

$$x_i = X_i$$

for all i.

Example 7: For the transform pair derived in Example 1,

$$f(a, b) = \overline{ab} \leftrightarrow F(A, B) = 1-AB$$

we can see that

a, b				B-exp	P-exp	P-exp $A=\frac{1}{2},\ B=\frac{1}{2}$
ab	$a\bar{b}$	$\bar{a}b$	$\bar{a}\bar{b}$			
0	0	0	0	0	0	0
0	0	0	1	$\bar{a}\bar{b}$	1-A-B+AB	1/4
0	0	1	0	$\bar{a}b$	B-AB	1/4
0	0	1	1	$\bar{a}$	1-A	1/2
0	1	0	0	$a\bar{b}$	A-AB	1/4
0	1	0	1	$\bar{b}$	1-B	1/2
0	1	1	0	$a\bar{b}+\bar{a}b$	A+B-2AB	1/2
0	1	1	1	$\bar{a}+\bar{b}$	1-AB	3/4
1	0	0	0	ab	AB	1/4
1	0	0	1	$ab+\bar{a}\bar{b}$	1-A-B+2AB	1/2
1	0	1	0	b	B	1/2
1	0	1	1	$\bar{a}+b$	1-A+AB	3/4
1	1	0	0	a	A	1/2
1	1	0	1	$a+\bar{b}$	1+A-AB	3/4
1	1	1	0	$a+b$	A+B-AB	3/4
1	1	1	1	1	1	1

Table 2. All B-exps and P-exps of two variables.

$f(0,0) = F(0,0) = 1$

$f(0,1) = F(0,1) = 1$

$f(1,0) = F(1,0) = 1$

$f(1,1) = F(1,1) = 0$

(End of example)

III. OPERATIONS WITH P-EXPS

This section is devoted to discussing operations which can be performed on P-exps, such as evaluation of the probabilities, and the inverse-transform. Of particular importance is the computational difficulty of manipulating these expressions, compared to that of B-exps.

In order to aid in the work described in this report, a computer program was written to obtain P-exps from logical network descriptions, and to manipulate the P-exps. The computer program will not be described in detail, except for some remarks on properties of P-exps that make them quite easy to handle.

3.1 COMPUTER REPRESENTATION OF P-EXPS

Each term of a P-exp, P_i, is easily stored as two parts: a "coefficient part," C_i, and a "value part," V_i. The coefficient part is a signed integer. The value part is a bit string where each bit position represents one of the function's variables, a 1 indicating that a particular variable is present in that term.

For example, for the transform pair,

$\overline{a}\,\overline{b} \leftrightarrow 1 - A - B + AB$

we would have:

$C_1 = 1, V_1 = 00$
$C_2 = -1, V_2 = 10$
$C_3 = -1, V_3 = 01$
$C_4 = 1, V_4 = 11$

The advantage of this representation is that when two terms, P_i and P_j, are multiplied, we have

$(P_i)(P_j)$
$= (C_i, V_i)(C_j, V_j)$
$= (C_iC_j, V_i \text{ OR } V_j)$

where "C_iC_j" represents ordinary integer multiplication of the coefficient parts, and "V_i OR V_j" represents a bit-wise ORing of the two bit strings representing the value parts. This automatically satisfies Theorem 2 (the IRR requirement).

3.2 COMPUTATIONAL COMPLEXITY

The Boolean operations NOT and AND form a complete set of Boolean connectives. These correspond to complementation and multiplication, respectively, of P-exps. These operations (Theorems 1 and 3) were implemented in the computer program that generates P-exps. We will discuss the relative complexities of performing these operations in the B- and P-domains.

Complementation of a P-exp of length L consists of cancelling the "1" term if it exists, else adding it, and changing the sign of each of the L C_i's. The complexity of this operation is obviously O(L).

Multiplying two P-exps with lengths L and M requires LM steps, because it is done as in ordinary polynomial multiplication. The difficulty is in cancelling and combining terms, since now there are LM terms. The most direct method would be to compare each term to every other, and when two value parts are identical, remove one term and add its coefficient part to the other's coefficient part. This takes care of both cancellation and combination. Unfortunately, this would require LM(LM-1)/2 or $O((LM)^2)$ steps. A better strategy is to sort the LM terms into either ascending or descending order on the basis of their value parts, and since all terms with the same value parts will be in consecutive positions, only an O(LM) algorithm is required to cancel and combine. Since computers are deterministic, the best sorting algorithm we can have (based on comparisons) is O(LMlogLM). The "Heap Sort" [44] achieves this lower bound, and was used in the P-exp software. The overall complexity, then, for the multiplication of two P-exps, using simple sorting techniques, is O(LMlogLM).

Although the Heap Sort allows us to reach the lower bound of O(LMlogLM) operations in reducing our list of LM P-terms, a different type of sort may allow us to reach a sorting and overall complexity of O(LM). In his Master's Thesis, Janus [34] describes the Distributive Partitioning Sort (DPS) which sorts N items in O(N) time when the items are uniformly distributed. The DPS was first published by Dobosiewicz [19].

Note that since we cannot predict, a priori, the number of terms resulting from the product of two P-exps, we cannot talk about the complexity of multiplying three P-exps, with lengths K, L, and M.

We can (for the product of two expressions) contrast the previously-discussed complexity with that of manipulating B-exps. Taking the product of two B-exps requires LM steps, resulting in LM terms, just as with P-exps, but then the result must be reduced according to the theorems in Table 1. Aside from simple reductions such as $x\bar{x} = 0$, this is probably impossible to perform in general because of pattern-recognition requirements. Simply taking the complement of a B-exp is harder than the NP-complete decision problem, "Satisfiability of Boolean Expressions" [25]. Applying DeMorgan's Theorem to a B-exp in SOP form containing L terms of K variables each, would require $O(K^L)$ multiplications, not counting reduction by the theorems in Table 1.

3.3 LENGTHS OF P-EXPRESSIONS

3.3.1 LENGTHS OF EXPRESSIONS FOR GENERAL FUNCTIONS

Consider a P-exp that is a function of N variables. The upper bound for the length of an expression can be determined as follows. Ignoring the coefficient parts of the terms, the number of unique value parts would be the number of unique binary words of N bits, so we could have at most 2^N terms. Is this upper bound achievable? Unfortunately, it is, and it appears to be the rule, rather than the exception. The all-zero minterm yields this worst case, for instance:

$$\bar{a}\bar{b}\bar{c} \leftrightarrow 1 - A - B - C + AB + AC + BC - ABC$$

and its complement is nearly worst case:

$$\overline{\bar{a}\bar{b}\bar{c}} = a + b + c \leftrightarrow A + B + C - AB - AC - BC + ABC$$

with a length 2^N-1, in the general case of N variables. In general, a single Boolean minterm with M complemented variables will yield a P-exp of 2^M terms.

For the three variable case, there are $2^{2^3} = 256$ possible functions. The worst case of length 8 occurs for 15 functions. There are 28 with length 7, 34 with length 6, 50 with length

5, 59 with length 4, 42 with length 3, 19 with length 2, and 9 with length 1. For the three variable case, then, the average P-exp length is 4.6196 terms.

For B-exps the worst case occurs when the Karnaugh Map displays a "checker-board" pattern, i.e., there are $2^N / 2 = 2^{N-1}$ minterms positioned such that no two can combine. Figure 3 shows such a pattern, with the DNF in summation notation listed below it. There are 32 minterms in this function of six variables, and because it is a checker-board pattern the DNF is also the minimal SOP. The P-exp for this function contains 63 terms and is also shown in the figure.

The average P-exp lengths for functions of two through seven variables have been determined by experimentation. The results are summarized in Table 3. For two and three variables the functions were generated exhaustively, but beyond that point it was necessary to resort to statistical methods, due to the amount of computation required.

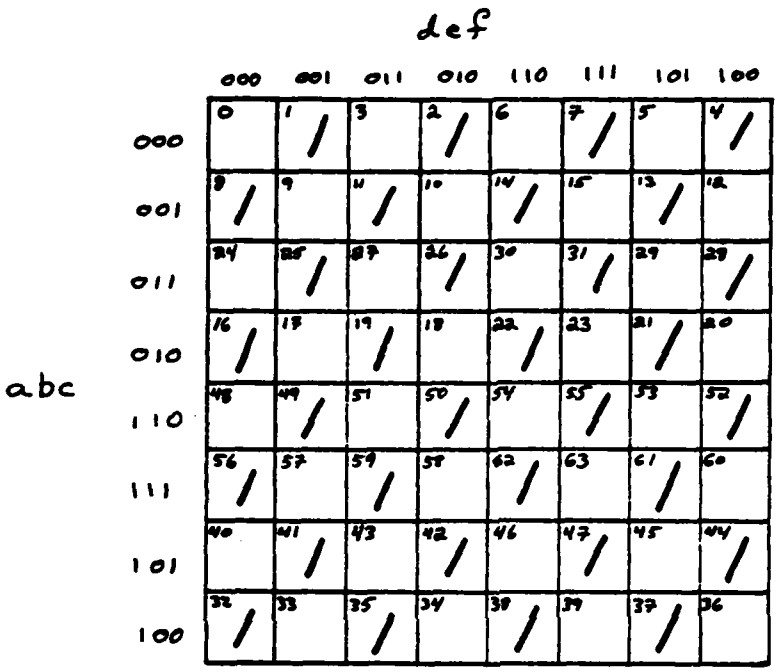
Logical functions were chosen randomly and were transformed into P-exps. For a function of N variables it was necessary to generate, using the set of 2^N minterms, functions that were uniformly distributed over the set of all functions of N variables. In other words, any given function could be generated with probability

$$1/(2^{2^N}) = 2^{-2^N}$$

The approach used was to generate "codewords" with a one-to-one mapping on to the set of all logical functions. For N variables there are 2^N minterms and $2**(2^N)$ possible functions. We used binary codewords of 2^N bits, where the bits were numbered from 0 to 2^N-1, representing the N-bit binary code for each vertex of an N-dimensional cube. For example, in a three variable case,

$$f(a, b, c) = \bar{a}c + ab$$
$$= \Sigma\ (1, 3, 6, 7)$$
$$= 01010011$$

Letting a particular minterm's probability of inclusion in a codeword be 1/2, a uniform distribution of codewords results. Constructing a function one minterm at a time (i.e., independently selecting each minterm with probability 1/2) would result in a Binomial distribution [43] of ones in the codewords. Letting y be the number of ones in a codeword, and p(y) be the probability of exactly y one bits,



B-exp = (1, 2, 4, 7, 8, 11, 13, 14,
16, 19, 21, 22, 25, 26, 28, 31,
32, 35, 37, 38, 41, 42, 44, 47,
49, 50, 52, 55, 56, 59, 61, 62)

P-exp = 16ABCDE + 16AbCDF + 16ABCEF + 4ABC + 16ABDEF + 4ABD
+ 4ABE + 4ABF + 16ACDEF + 4ACD + 4ACE + 4ACF + 4ADE + 4ADF
+ 4AEF + A + 16BCDEF + 4bCD + 4bCE + 4bCF + 4bDE + 4BDF
+ 4BEF + B + 4CDE + 4CDF + 4CEF + C + 4DEF + D + E + F
- 32ABCDEF - 8ABCD - 8ABCE - 8ABCF - 8ABDE - 8ABDF - 8ABEF
- 2AB - 8ACDE - 8ACDF - 8ACEF - 2AC - 8ADEF - 2AD - 2AE
- 2AF - 8BCDE - 8BCDF - 8BCEF - 2BC - 8BDEF - 2BD - 2BE
- 2BF - 8CDEF - 2CD - 2CE - 2CF - 2DE - 2DF - 2EF

Figure 3. Karnaugh Map of Six Variables Filled With Checker-Board Pattern.

N = #variables	2^{2^N} = #functions	Ave. P-exp length* exact	Ave. P-exp length* 500 random functions	x^N (using random func.)
2	16	2.2667	2.326	1.525^2
3	256	4.6196	4.632	1.667^3
4	65536	?	10.06	1.781^4
5	4.295×10^9	?	21.436	1.846^5
6	1.845×10^{19}	?	45.168	1.887^6
7	3.403×10^{38}	?	95.904	1.919^7

*Not including the all-zero functions

Table 3. P-exp Lengths as Function of Number of Variables.

$$p(y) = \binom{2^N}{y} p^y q^{2^N - y}$$

$$y = 0, 1, \ldots 2^N$$

and since $p = q = 1/2$,

$$p(y) = \binom{2^N}{y} (\tfrac{1}{2})^y (\tfrac{1}{2})^{2^N - y}$$

$$= \binom{2^N}{y} (\tfrac{1}{2})^{2^N}$$

$$= \frac{\binom{2^N}{y}}{2^{2^N}}$$

which is exactly the probability of a codeword of length 2^N having a Hamming Weight of y, out of a possible $2^{**}(2^N)$ codewords.

The average P-exp lengths in Table 3 were the average lengths of 500 randomly-generated functions of N variables. Included in these averages, of course, are functions of fewer than N variables, e.g., f (a, b, c) = a + b. The column labeled "x^N" indicates the average growth in complexity as a function of N. We know that it has an achievable upper bound of 2^N. We note, to our dismay, that x does appear to approach 2. Apparently, the number of functions that have P-exp lengths of 2^N or close to it, as N increases, contributes so overwhelmingly to the total that the average length asymptotically approaches the worst case.

3.3.2 LENGTHS OF EXPRESSIONS FOR UNATE FUNCTIONS

Unate functions [42] are logical functions which, in their unique minimal form, have no variable in both its complemented and uncomplemented form. Without loss of generality, we will discuss only monotonically increasing unate functions, i.e., those which have only uncomplemented variables.

Unate functions are important to our discussion because of their usefulness in certain kinds of problems, e.g., the reliability evaluation of communications networks. The previous

exposition on the "average" lengths of general P-exps showed that there is an exponential growth in their lengths, as a function of the number of input variables. We will show that unate functions have P-exps that increase in length at a lower rate than those for general functions.

Previously, we expressed each logical function of N variables as a codeword of 2^N bits, where each bit represented a minterm of the function. There are then $2**(2^N)$ possible codewords, which equals the possible number of logical functions of N variables. For our empirical experiment we generated a uniform distribution over these codewords, and P-transformed these.

There is a many-to-one mapping of the codeword space representing arbitrary logical functions on to the codeword space representing only unate functions. Any non-unate function can be converted to a unate function by means of a minterm covering relationship. McNaughton [42] defined an ordering of the vertices of an N-cube (for a function of N variables) such that for two vertices, x and y, where $x = (x_1, x_2, \ldots x_N)$ and $y = (y_1, y_2, \ldots y_N)$, then $x \leq y$ if and only if $x_i \leq y_i$ for all i. X is then said to cover y. For example, $(1010) \leq (1110)$, but (1010) and (1100) are incomparable. Unate functions possess the property that if x is a true vertex (a minterm) then any vertex that covers x is also a minterm (see McNaughton [42, Theorem 2.1]).

This fact is used here to transform non-unate functions into unate functions. The uniformly-distributed codewords of the previous section were converted into unate functions by performing an extra pass, causing every vertex that covered an existing minterm to become a minterm also. An alternative method would have been to use the false vertex method, i.e., starting with the "largest vertices" and cause every vertex that is covered by x (where $f(x) = 0$) not to be a minterm.

The results of this experiment are summarized in Table 4. The all-zero and all-one functions were omitted from consideration, the first because it is a trivial case, and the second because it is not only trivial but would have misleadingly cut the average P-exp length nearly in half. Fifty percent of the uniformly-distributed random codewords include the zero minterm, which is worst-case for the general functions, but in the unate case results in a P-exp that is exactly equal to 1 (because all vertices cover that minterm).

N = #variables	#unique functions*	Ave. P-exp length* using 500 functions** chosen randomly	x^N	Ave. len. for unate / Ave. len. for nonunate
2	4	1.571	1.223^2	0.6754
3	18	3.032	1.447^3	0.6546
4	166	6.18	1.577^4	0.6143
5	?	12.552	1.659^5	0.5856
6	?	26.01	1.721^6	0.5759
7	?	56.484	1.779^7	0.5890

*Not including all-zero or all-one functions

**Exact lengths for 2 and 3 variables

Table 4. P-exp Lengths for Monotonically-Increasing Unate Functions, as Function of Number of Variables.

In Table 4, note that the average P-exp length appears to grow less rapidly than for the general case. The last column shows that the ratio of the length of unate versus nonunate functions appears to approach about 60% or less.

3.4 CALCULATING PROBABILITIES

One of the major applications of P-exps is to obtain signal probabilities. Once a P-exp has been obtained the calculation of a numerical probability is readily done. This can be done in time proportional to the number of terms in the P-exp.

Example 8: Consider the following transform pair:

ab + cde <--> AB + CDE - ABCDE

If we assign the probabilities A = B = C = D = E = 1/2, then we can evaluate the P-exp as follows:

$$0.5^2 + 0.5^3 - 0.5^5$$
$$= 0.25 + 0.125 - 0.03125$$
$$= 0.34375$$

This again is the probability that the B-exp has the value 1.

For this particular B-exp, this result is easily verified either from a truth table, or by using the fact that it is the union of two independent events:

$$0.5^2 + 0.5^3 - 0.5^2 0.5^3$$
$$= 0.34375$$

(End of example)

3.5 SOLVING P-EXPS

By solving a P-exp we mean to find a set of input conditions that cause the P-exp to equal either 0.0 or 1.0 as desired. We will refer to these problems as "solving for 0" and "solving for 1." Here, we use the fact that if the input probabilities are set to exactly 0 and 1 then the output will equal the output of the original B-exp (Theorem 6). Simply determining

whether a B-exp in SOP form has a 0 solution (or whether a B-exp in POS has a 1 solution) is the NP-complete decision problem Satisfiability [25]. We will show that a P-exp, on the other hand, can be solved in O(N) time, where N is the number of variables. In the P-domain the NP-complete nature of the Satisfiability problem does not hold since a P-exp with no 0 solution will identically equal 1, and a P-exp with no 1 solution will identically equal 0. Thus, going to the P-domain reduces the complexity of this problem. However, the complexity of going back and forth would have to be considered.

A simple algorithm for solving P-exps in O(N) time will now be described. Let the P-exp be F(A, B, C,...), and we will refer to A as the first variable, B as the second, etc. B is the next variable with respect to A. If we wish to solve for 1, then:

Step 1: Let all input variable probabilities equal 1/2. This will be called the unassigned state.

Step 2: Assign to the first (or "next") unassigned variable the value 1.0, evaluate the P-exp, and let "P_1" represent this probability. Then assign to the same variable the value 0.0, and re-evaluate the P-exp, letting "P_0" represent this probability.

Step 3: If $P_1 \geq P_0$, then permanently assign to the variable from Step 2 the value 1.0. Otherwise (meaning that $P_1 < P_0$), permanently assign to the variable the value 0.0.

Step 4: If there are unassigned variables remaining, go to Step 2. Otherwise, stop - the values of the assigned variables constitute a 1.0 solution.

Note that if a 0.0 solution is required, then we need only change the directions of the inequalities in Step 3.

Once a variable is permanently assigned it is never changed, and since each variable is eventually checked, the number of loops in the algorithm equals the number of variables. Therefore, time is O(N).

In this algorithm no provision is made for stopping if the P-exp identically equals the required solution before all variables are assigned (making the rest of the variables don't-cares). A more sophisticated algorithm could loop through Steps 2 and 3, within each major loop, to find the single best variable assignment to increase the "yield," and thus the

probability of being able to reach the solution with a minimum number of permanent assignments (maximizing don't-cares). Neither of these strategies is necessary to guarantee the O(N) performance, so we will not dwell further on them. Also, neither of these strategies guarantees a minimal solution (where a minimal solution would be defined as a solution such that no other solution exists that contains more don't cares).

Example 9: An example of the algorithm is given in Figure 4 for the transform pair

$$\bar{a}bc\bar{d}\bar{e} + ace \leftrightarrow ABD + ACE + ABCDE - ABCD - ABDE$$

The answer obtained (11111) is immediately verified by inspection of the B-exp.

(End of example)

Two observations must be made of the "tree" in Figure 4. First, at each point where a variable assignment is made, a "degree of freedom" is lost, meaning that the P-exp becomes a function of one less variable. The probabilities become conditional, which is why the probabilities increase as the algorithm is applied. Second, although the algorithm calls for the highest probability path to be followed, actually any non-zero probability branch can be taken (when solving for 1.0, of course).

3.6 INVERSE P-TRANSFORM

So far we have discussed the transformation from B- to P-exp. In theory it should be possible to apply Theorems 1, 3, and 4 in the opposite direction, but this is impractical.

We will demonstrate the difficulty of the inverse P-transform when using the given theorems:

Example 10: The simple P-exp,

$$AB + CDE - ABCDE$$

is easily inverse-transformed by a single application of Theorem 4, yielding,

$$ab + cde$$

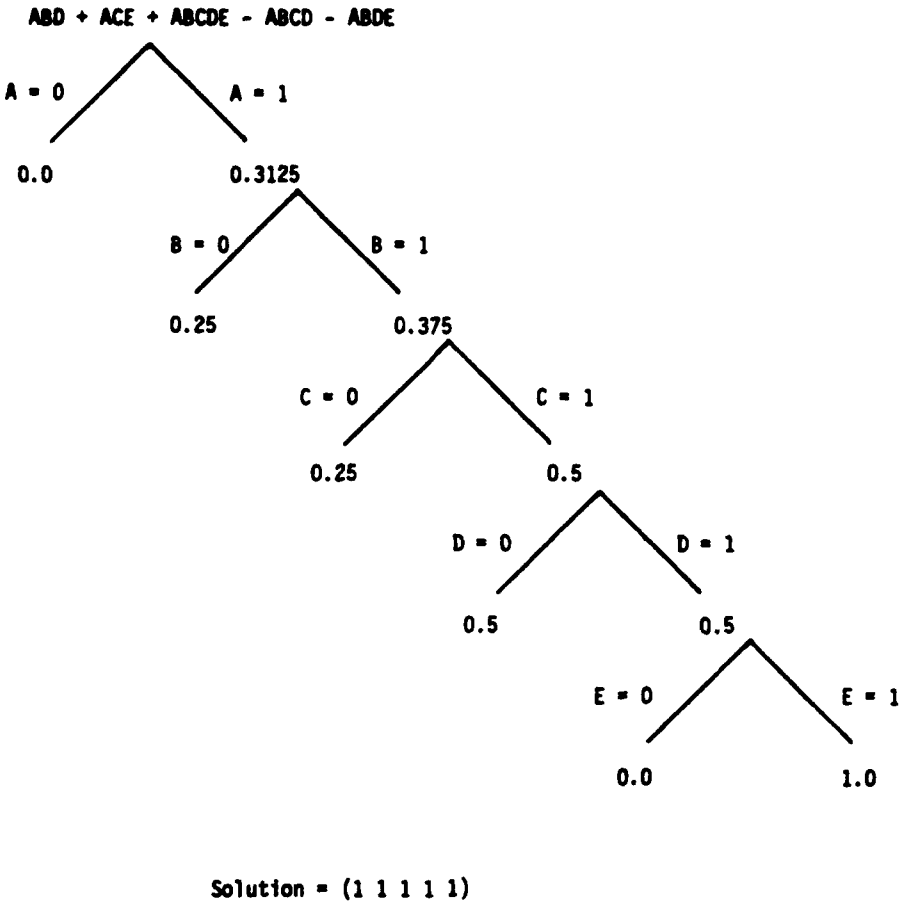


Figure 4. Solving a P-exp for 1.

(End of example)

Example 11: A less tractable P-exp is the following:

$$AB + C - AC$$

unless we realize that the last two terms can be grouped, and C can be factored out,

$$\begin{aligned} & AB + C - AC \\ &= AB + C(1 - A) \\ &= AB + \overline{A}C \end{aligned}$$

Applying Theorem 4, the B-exp is found to be:

$$ab + \overline{a}c$$

(End of example)

A P-exp which is probably beyond direct manual methods is:

$$1 + ABC + BC - AB - C \qquad (6)$$

We will show two methods of finding a B-exp that corresponds to a given P-exp.

Method 1: As a consequence of Theorem 6, a P-exp takes exactly the values 0 and 1 when its inputs take these same values. Thus, we can recreate the original list of minterms by applying all 2^N combinations of N input variables, in each case by evaluating the expression and noting which of the combinations have 1 as the output. These input combinations represent the minterms. The only way this technique could be made easier is to evaluate the P-exp with equiprobable inputs beforehand, in order to determine whether the list of maxterms is shorter to store than the list of minterms. The minimum and average run times equal the maximum, which is 2^N.

Example 12: An example of this is shown in Figure 5. This shows the inverse transformation of Eq. 6, by evaluating and summing its terms.

(End of example)

a b c	1 + ABC + BC - AB - C					
0 0 0	1					1
0 0 1	1				-1	0
0 1 0	1					1
0 1 1	1		+1		-1	1
1 0 0	1					1
1 0 1	1				-1	0
1 1 0	1			-1		0
1 1 1	1	+1	+1	-1	-1	1

$$\sum (0, 2, 3, 4, 7) = a\ b\ c + \overline{a}\ b + \overline{b}\ \overline{c}$$

Figure 5. Inverse P-Transform, Method 1.

We can get better performance than this by applying the Shannon Expansion Theorem (see Table 1), which leads to the second method:

Method 2: A more organized approach than the exhaustive search is to build a decision tree, where at each node two branches are formed as in Figure 6. New branches and nodes are formed until a node is created where the P-exp equals 0 or 1. Now go back up the tree, forming B-exps in accordance with the Shannon Expansion Theorem.

Example 13: Figure 7 shows the inverse-transformation of Eq. 6 by Method 2. The result obtained is equivalent to that obtained by Method 1. To verify the result,

$$f = abc + \bar{a}b + \bar{b}\bar{c}$$

Applying the P-transform:

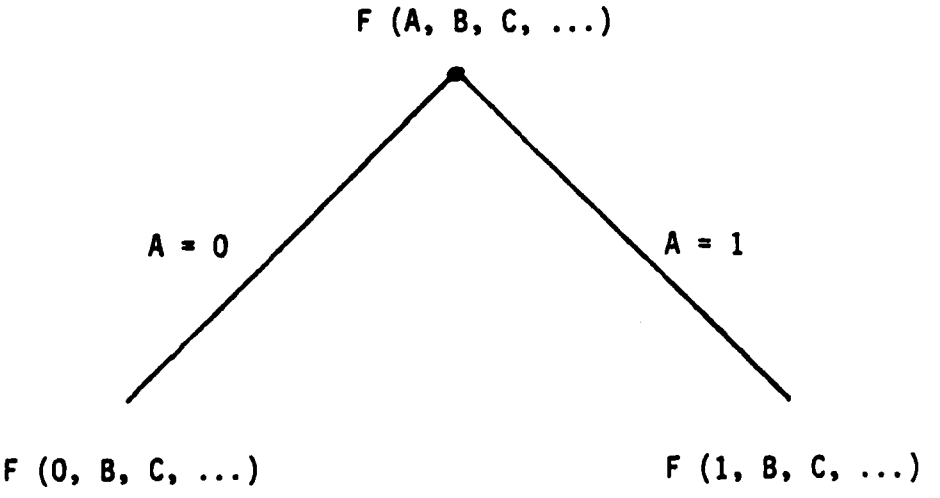
$$\begin{aligned} F &= ABC + \bar{A}B + \bar{B}\bar{C} \\ &= ABC + B - AB + 1 - B - C + BC \\ &= 1 + ABC + BC - AB - C \end{aligned}$$

which is identical to Eq. 6. Therefore,

$$abc + \bar{a}b + \bar{b}\bar{c} \leftrightarrow 1 + ABC + BC - AB - C$$

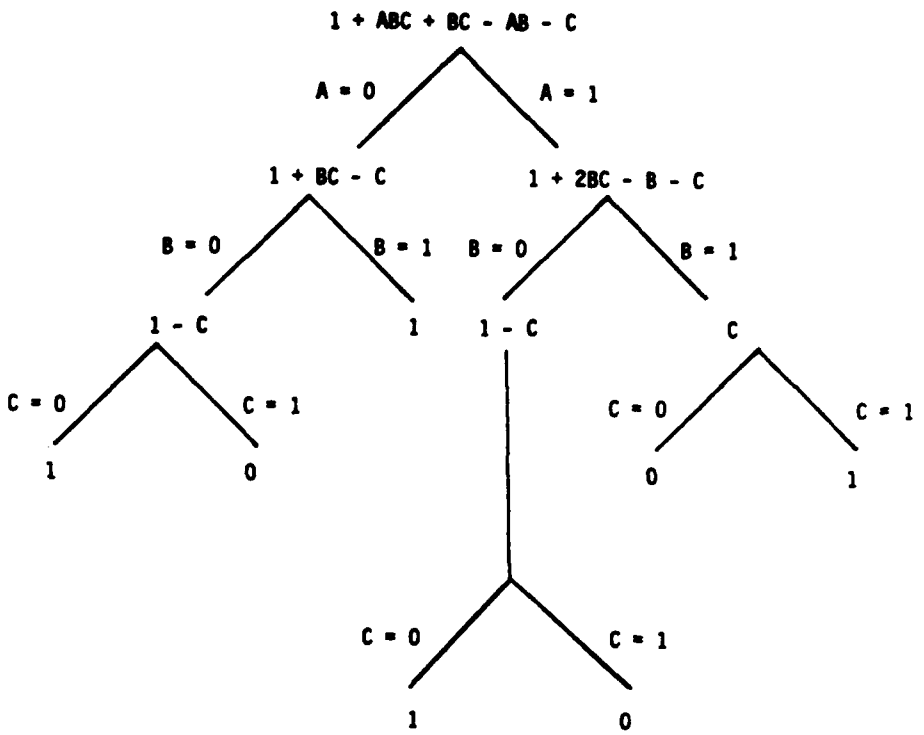
(End of example)

Remarks on the complexity of Method 2 are in order. Worst-case time complexity is $O(2^N)$, since we must potentially follow every branch to the full limit of N levels (where N is the number of independent variables). Storage complexity is not bad, except for storing the B-exp as it is generated (which, admittedly, is generated in an inconvenient form). If this operation is implemented recursively, then the deepest stack descent will be N levels, and only one complete branch is "stored" at a time. Only one copy of the P-exp is required, since at each level it is simply evaluated with different input conditions (not necessarily modified to remove variables). It is possible to have the B-exp incrementally printed at each node shown in Figure 7 (the first B-exp, not the final, simplified version).



$$f(a, b, c, \ldots) = a \cdot f(1, b, c, \ldots) + \bar{a} \cdot f(0, b, c, \ldots)$$

Figure 6. Shannon Expansion Theorem.



$$\bar{a}\,(\bar{b}\,(\bar{c}\cdot 1 + c\cdot 0) + b\cdot 1)$$
$$+\,a\,(\bar{b}\,(\bar{c}\cdot 1 + c\cdot 0) + b\,(\bar{c}\cdot 0 + c\cdot 1))$$

$$= \bar{a}\,(\bar{b}\,\bar{c} + b) + a\,(\bar{b}\,\bar{c} + b\,c)$$
$$= \bar{a}\,\bar{b}\,\bar{c} + \bar{a}\,b + a\,\bar{b}\,\bar{c} + a\,b\,c$$
$$= \bar{b}\,\bar{c} + \bar{a}\,b + a\,b\,c$$

Figure 7. Inverse P-Transform, Method 2.

This algorithm does not stop until reaching 0 or 1 at the end of a branch. A more sophisticated algorithm could recognize when the P-exp at a node forms a simple term, such as BCD, and then stop, because, for example,

bcd <--> BCD

or it could recognize other simple P-exp constructs, such as

$\overline{c}$ <--> 1 - C

or

$\overline{bcd}$ <--> 1 - BCD

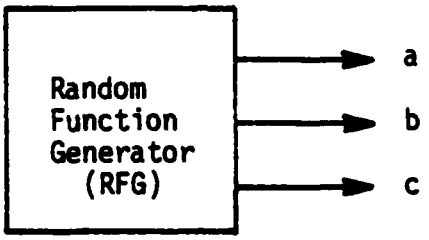
This approach might be difficult to reconcile with having only a single copy of the P-exp. On the other hand, at most N P-exps would have to be stored at any time, and the lengths of the expressions diminish in size quite rapidly.

3.7 RANDOM FUNCTION GENERATORS

P-exps can be used to aid in the construction of a Random Function Generator, or RFG, which is a set of logical functions resulting in probabilistic outputs having certain desired properties. The random input variables to a P-exp can individually have any value on the closed interval [0.0, 1.0], but, by definition, they must be statistically independent. An RFG is a multiple-input, multiple-output logical function (modeling a combinational network) where the output probabilities satisfy given requirements of mutual exclusiveness, or conditional probabilities. Although examples will not be given here, RFGs can be used for the solution of word problems, or for managing complex relationships of joint and conditional probabilities.

3.7.1 RFG WITH MUTUALLY-EXCLUSIVE OUTPUTS

The simplest RFG results when the outputs are required to be mutually-exclusive. Consider the general three-output RFG, with outputs a, b, c, which are to be described by "mutually-exclusive" P-exps A, B, C, as shown in Figure 8. We will let P(a), P(b), and P(c) be the numerical output probabilities required. The output probabilities may sum to less than 1.0. Figure 9 shows an implementation of this, with logical variables a, b, and c generated as



$$P(a, b) = 0$$
$$P(a, c) = 0$$
$$P(b, c) = 0$$

} mutually exclusive outputs

$$0 < P(a) + P(b) + P(c) \leq 1$$

Figure 8. Three-Output Mutually-Exclusive Random Function Generator (RFG).

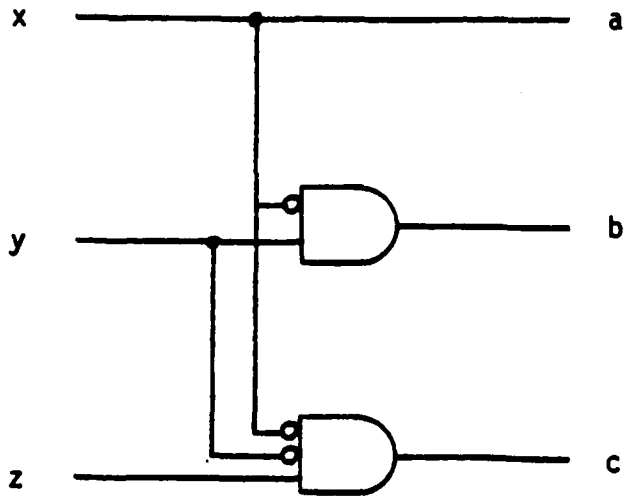


Figure 9. Construction of the Random Function Generator of Figure 8.

functions of the statistically independent probabilistic variables X, Y, and Z. The value of X is clearly P(a). For Y, we know that,

$$P(b) = P(\overline{x}, y)$$

or,

$$P(b) = (1 - X)Y$$

so that,

$$Y = P(b)/(1-X)$$
$$= P(b)/(1-P(a))$$

For Z, we perform a similar manipulation,

$$P(c) = P(\overline{x}, \overline{y}, z)$$
$$= (1-X)(1-Y)Z$$
$$= \left[1 - P(a) - \frac{P(b)(1-P(a))}{1-P(a)}\right]Z$$
$$= (1 - P(a) - P(b))Z$$

$$Z = \frac{P(c)}{1 - P(a) - P(b)}$$

Example 14: Consider an RFG with three mutually-exclusive outputs where P(a) = 1/3, P(b) = 1/4, and P(c) = 1/5. From above, X = 1/3, Y = (1/4)/(1-1/3) = 3/8, and Z = (1/5)/(1-1/3-1/4) = 12/25. From the network in Figure 9 realizing an RFG, we can write three P-exps:

$$A = X$$
$$B = (1-X)Y = Y - XY$$
$$C = (1-X)(1-Y)Z = Z - XZ - YZ + XYZ$$

Multiplying, or ANDing, any two of these P-exps results in 0, demonstrating mutual exclusiveness for this general three-output case, and after substitution of the numerical values for X, Y, and Z that were obtained above, we find that

A = 1/3
B = 1/4
C = 1/5

which are exactly the desired probabilities.

(End of example)

It is clear that using this method one can generate any set of mutually-exclusive outputs with arbitrary precision. The maximum number of statistically independent input variables equals the number of outputs, although if the sum of the output probabilities equals 1.0 then the last input variable can be omitted.

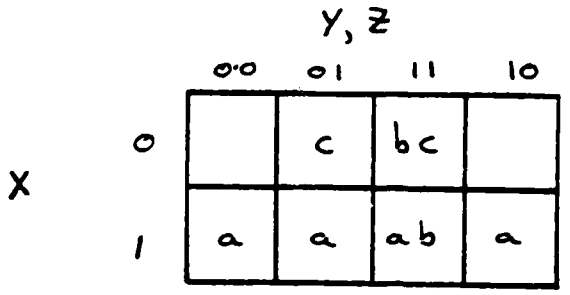
3.7.2 RFG WITH NON-MUTUALLY-EXCLUSIVE OUTPUTS

A more general technique is given by Klir [35]. His approach is not restricted to outputs which are mutually-exclusive, but has the disadvantage of being able to generate only output probabilities that are negative powers of 2.

Example 15: Klir's method will be demonstrated. Consider a three-output RFG, where the desired conditions are:

P(a) = 1/2
P(b) = 1/4
P(c) = 1/4
P(a, b) = 1/8
P(a, c) = 0
P(b, c) = 1/8

The output variables are no longer mutually-exclusive. The P-exps A, B, and C will be directly derived from a Karnaugh Map, and, generally, can be arbitrarily chosen. The smallest "unit" of probability required in this problem is 1/8 because all other probabilities are 1/8. Therefore, three statistically independent input variables with values exactly equal to 1/2 are required, because $(1/2)^3 = 1/8$. Figure 10 shows a Karnaugh Map assignment which satisfies the requirements, the the B- and P-exps, and the realization. ANDing of the pairs of expressions, and substitution of X = Y = Z = 1/2 results in the desired probabilities.



$a = x \longleftrightarrow A = X$

$b = y\ z \longleftrightarrow B = YZ$

$c = \overline{x}\ z \longleftrightarrow C = Z - XZ$

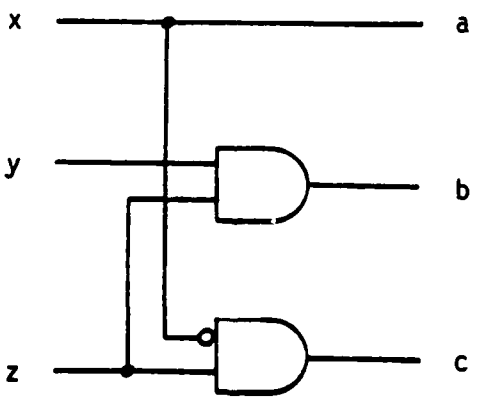


Figure 10. Random Function Generator With Non-Mutually-Exclusive Outputs.

(End of example)

The disadvantage of this method is apparent when probabilities such as 1/3 or 1/5 are called for. These values must be approximated. For example, the binary representation for 1/3 is the continued binary fraction

0.01010101...

Using eight binary digits of precision yields a decimal digit value that is exactly 0.33203125 (which is accurate within 0.4%). However, this requires a Karnaugh Map of eight variables. A five variable map would give an approximation of 0.3125 (accurate within 6.25%) and six variables would give 0.328125 (1.56%). In this approach there is, additionally, the human problem of assigning the functions so as to achieve the desired relationships. However, this is really no more complicated than drawing Venn Diagrams to scale on graph paper, if minimization of the logical functions is not attempted. Further, the "area" required for each output variable need not be composed of a single cube on the map, although these examples did not demonstrate this option. An output variable may be made up of several cubes (covering the total number of required minterms), and the RFG would use AND/OR or NAND/NAND logic to sum these areas.

IV. APPLICATION OF P-EXPS TO TESTING OF COMBINATIONAL NETWORKS

In this section we apply P-exps to the probabilistic analysis of test generation for combinational digital networks. This problem has been addressed in the literature [1-5,53], in some cases using similar tools [45-47]. We will deal with this problem in considerably more depth, and the techniques will be applicable to general combinational networks, rather than the restricted types that are generally studied.

4.1 THE TESTING PROBLEM

Digital networks are tested in order to detect faults. Here, a fault will mean a single, permanent, stuck-at-zero or stuck-at-one (SA0, SA1) condition of a logical signal. A fault is a simplification of the actual "failure mechanisms" of circuits, such as oxide opens or metalization defects in the case of a microcircuit. A failure is the instance of a fault causing the output to change from the expected logic level.

Test generation techniques may be described broadly as deterministic or statistical. The distinction is not as sharp as we would like when trying to classify actual test generator implementations, but that is not a great difficulty here. A discussion of these two approaches is necessary before describing the P-exp approach.

We point out that the term testing confidence level means different things in the deterministic and statistical senses. In the former case, it is used to mean the percentage of detected faults to the total number of possible faults, while in the latter case it is a percentage of our "confidence" that the network is entirely fault-free. Depending on the definitions of the fault equivalence classes, these may mean the same thing.

4.2 DETERMINISTIC TEST GENERATION

A deterministic test is one that is intended to detect a specific fault or set of faults. One way of applying a deterministic test generation algorithm would be to list all of the possible faults in a network, and then generate a test for each fault. This would be an inefficient approach because, in general, a single test detects a multiplicity of faults. Commercial Digital Automatic Test Pattern Generators (DATPGs) apply more cost-effective methods.

Techniques exist for finding the faults that are detected by a particular test. An example of a fault simulation or grading technique is given in a landmark paper by Armstrong [8], and today even more efficient techniques are known.

With a fault simulator available, most commercial DATPGs use a multipass strategy:

Step 1: Select a threshold as an intermediate goal, perhaps 10% of the remaining number of undetected faults (initially, all of the faults in the network are "remaining").

Step 2: Select several of these undetected faults and generate tests for them, attempting only to detect up to the given threshold. A "quick and dirty" meaure is provided for estimating when the threshold has been reached.

Step 3: Grade the actual fault detection using a fault simulator. Usually more faults are detected than just the selected ones.

Step 4: If the cumulative fault detection level is greater than a given level (such as 95%), stop. Otherwise, go back to Step 2 and repeat the process.

The above fault detection experiment is not guaranteed to terminate, as some faults may not be detectable, or may be detectable only by accident by the particular deterministic test generation algorithm being used. The effect of serendipity in Step 3 (the detection of extra faults) causes most deterministic test strategies to behave like statistical test generators during the early stages. The reason that this is done in several small "chunks" is to take advantage of the accidental fault detection. In later stages the faults are generally "hunted down" and detected one at a time.

4.2.1 FANOUT-FREE NETWORKS

For the special case of fanout-free networks, the Berger-Kohavi Algorithm (BKA) [12] was shown by its authors to produce an optimal set of tests for every single stuck-at fault, and can be shown to detect every multiple fault as well [29]. The BKA is optimal in the sense that no smaller test set can be generated which detects all the faults.

To show the performance of the BKA, consider a network consisting of identical n-input NAND gates forming a tree-type network of L levels, where every level is completely filled. There will then be $n^L = N$ primary inputs. In [12] a counting algorithm is given for the number of tests required, and using that algorithm the BKA can be shown to require exactly

$$(\sqrt{n} + 1/\sqrt{n})(\sqrt{N})$$

tests for a network with an odd number of levels, and exactly

$$2\sqrt{N}$$

tests for a network with an even number of levels. The number of tests for this type of network is therefore $O(\sqrt{N})$. While this is exponential in L, it is much better than linear in N. It should be noted that for a network consisting of exactly one gate with fan-in of n, so that $N = n^L = n^1 = n$, then the expression given above for the number of tests for a network with an odd number of levels reduces to N+1, which is O(N).

Another complexity measure can be obtained based on the gate count, G. The tree network used here has

$$G = 1 + n + n^2 + n^3 + \ldots + n^{L-1}$$

$$= (1-n^L) / (1-n)$$

gates. Since $N = n^L$, this is then

$$G = \frac{n}{n-1} N - \frac{n}{n-1}$$

which is clearly linear in N, so the number of tests for a multilevel tree-type network can be considered to be $O(\sqrt{G})$ as well.

4.2.2 THE D-ALGORITHM

For general combinational networks that include fanout, the D-Algorithm [39,50] is the best known technique for test generation. Enhancements have been made to improve certain features, such as the consistency operation in the A-Algorithm [6], the forward drive or D-propagation [17], or in application to non-gate primitive elements [15,40-41]. For our purposes we shall use only part of the sophistication of the D-Algorithm, as an aid to demonstrating the path sensitization problem in deterministic test generation.

Let us review some aspects of the D-Algorithm. In this algorithm D represents a signal that takes the value 1 in the good network, which becomes a 0 in the presence of a fault. A $\overline{D}$ represents the opposite condition. Figure 11 depicts the propagation of a five-valued system through a NAND gate (0, 1, D, $\overline{D}$, and X, where X represents a "don't know" condition that could be any of the other four values). After selecting the site of a fault, there are three steps. The first is activation of the fault. This consists of selecting the proper value at the site of the fault (e.g., for a SA0 fault we want a 1, or D placed there). The second step is sensitization or propagation, which makes the output of the network sensitive to the logical value at the fault site. One may think of the value as being propagated through the logic by selection of the proper set of enabling values. The third step is the line justification or consistency operation. This is the most difficult step, which is the process of selecting the correct primary input values to cause the desired internal conditions to occur.

We will usually group the activation and propagation operations together. The P-exp approach will be applied when we have the activation and propagation conditions given to us, and we will apply our technique to the harder task of line justification.

An example of the D-Algorithm is given in Figure 12. The site of a suspected SA0 fault is assigned the value D (which will be 1 in the good network). The D is propagated in accordance with the table in Figure 11, becoming a $\overline{D}$ and finally a D at the primary output. The SA0 fault will thus change the output from a 1 to a 0. In the consistency operation the primary input values were derived that will put the D at the fault site, and cause the propagating values to appear at the other inputs of the gates.

This example showed single path, or one-dimensional, sensitization. Double path, or two dimensional, sensitization is required when a D or $\overline{D}$ value cannot be propagated using a 1 (for a NAND or AND gate), but only by using another D or $\overline{D}$ value. An example of this is shown in Figure 13. Even more complicated constructions may be necessary in order to

a	b	c
0	X	1
X	0	1
1	D	$\overline{D}$
1	$\overline{D}$	D
D	1	$\overline{D}$
$\overline{D}$	1	D
D	D	$\overline{D}$
$\overline{D}$	$\overline{D}$	D
D	$\overline{D}$	1
$\overline{D}$	D	1

X = don't care

Figure 11. D-Propagation Through a NAND Gate.

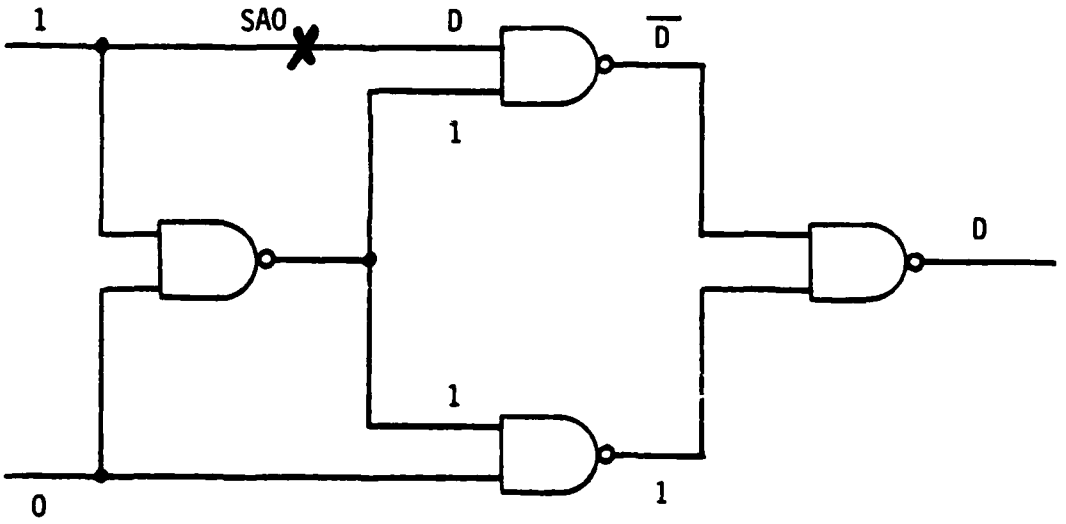


Figure 12. D-Algorithm, Single Path Sensitization.

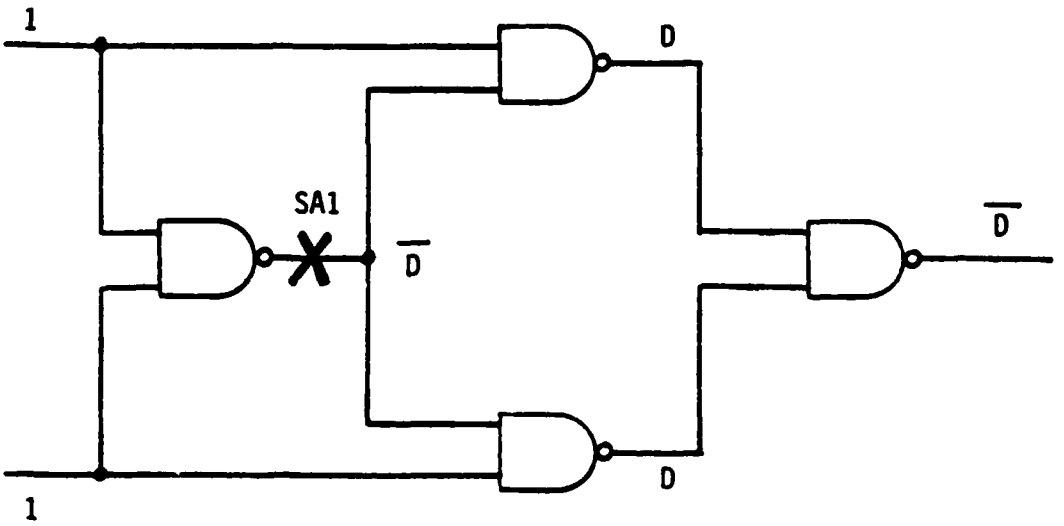


Figure 13. D-Algorithm, Double Path Sensitization.

obtain a test. The D-Algorithm tries all single paths before double paths, all double paths before triple paths, and so on.

For any given fault that is detectable, the D-Algorithm need generate only one test vector. An upper bound, then, for the number of tests is simply the number of possible faults, which is generally a small multiple of the number of gates in the network (using the single fault assumption). As discussed earlier, this upper bound should never be reached because many faults are detected at a time. It is not the test length, but rather the computational effort required to find tests that causes the D-Algorithm, or any deterministic testing algorithm for general networks, to be NP-complete. Ibarra and Sahni [33] showed that several important problems in the testing of general networks are NP-complete, including the decision problem, "Can a fault in a particular input line x_i be detected by input-output experiments?" Fujiwara and Toida [23] have shown that test generation for monotone/unate networks, a simpler class of networks, is still NP-complete.

In spite of the NP-completeness of this type of test generation, the performance of commercial automatic test pattern generators is generally not bad, in practice. Goel [27] has presented empirical evidence to show, for his PODEM-X Algorithm (similar to the D-Algorithm), that test pattern generation costs grow as G^2, and fault simulation costs grow as G^2 to G^3, where G is the gate count.

4.3 STATISTICAL TEST GENERATION

We may place almost any technique that is not strictly deterministic under this category. These techniques characteristically involve the assignment of probabilities. The sort of question usually asked is: "Given a set of M random vectors has been applied and no failure is observed, what is the probability that the network is fault-free?" Random testing is generally considered to be exponential in difficulty because, for complete testing, a network with N input lines would require 2^N unique test vectors.

A number of approaches exist. Test generation may be performed with replacement (the same test conditions may be generated again and again), or without replacement (every test condition is unique). Input signal probabilities may be equiprobable or may be adjusted to increase the "yield." One may find the number of tests required to find a single specific fault, or an upper bound on the time required to find all faults.

Some Random Test Generation Approaches

Parker and McCluskey [46-48] used P-exps to analyze the variable input probability problem. However, they restrict themselves to an approach they call bundling, which means that they assign the same variable probability to all of the inputs. They gave as an example an AND/OR network that realized the P-transform pair

$$z = abcde + fghi \leftrightarrow Z = ABCDE + FGHI - ABCDEFGHI$$

where z is the output. They assign the same probability, X, to all input signal lines, resulting in a "bundled" P-exp:

$$Z = X^4 + X^5 - X^9$$

where Z and X are numeric values. The authors show that, in the case where they wish to detect z SA0, the usual assignment of X = 1/2 (resulting in Z = 0.0918) provides a poor SA0 test, while as X approaches 1.0 the testing quality (for that one fault) improves. It is obvious that the more likely a test for z SA0 is, the less likely the test for z SA1 will be. The best tradeoff for detecting both faults would be to cause Z = 1/2, which occurs when X = 0.75893. However, this tells us nothing about the effectiveness of detecting other faults in the network.

In non-trivial networks bundling could defeat the testing goals. Schnurmann, Lindbloom, and Carpenter [54] describe test effectiveness in terms of the switching activity count of internal nodes in a sequential network. For a specific case they note that, "In terms of the final fault-detection patterns, overactive resets are undesirable." Similarly, for a device such as a microprocessor the clock should have a high probability of switching on every test cycle (adding a new probabilistic dimension), and a combinational network with an "enable" input should have a high probability of being enabled, regardless of what the other inputs are doing.

Agrawal and Agrawal [1-3] have applied random testing techniques to fanout-free networks. They have expressed the probability of input faults being propagated to the output in terms of the probability of the faulty signal being propagated at each level. Input faults are the only ones considered, because Hayes [30] has shown that any internal stuck-at fault in a fanout-free network is equivalent to a fault (single or multiple) at the inputs only. Agrawal and Agrawal restrict their discussion to tree-type networks composed only of n-input NAND

gates. Their strategy is to compute the necessary number of vectors to achieve a given probability of detection. In the two-input NAND case, their required number of vectors is exponential in both L and N. Therefore the performance of this approach for such a simple network is much worse than the $O(\sqrt{N})$ performance of the deterministic algorithm BKA, as shown earlier.

Agrawal [4], drawing on earlier work published in [1], gave a short tutorial on "when to use random testing." As before, results are derived for tree-type networks composed of L levels of n-input NAND gates. His approach is to bundle all input probabilities to an "optimum" value p that satisfies the equation

$$p = 1 - p^n$$

which causes the signal probability at each level to be the same value. He then calculates the number of test vectors required to sensitize an input fault through the L levels to the output. His goal is to show that if the number of test vectors required using this calculation is less than the 2^N exhaustive test patterns, then random testing is recommended. Otherwise, either exhaustive testing or deterministic test generation should be performed.

Agrawal [5] discussed testing effectiveness in terms of information flow through partitions of the network. His only example was the somewhat trivial case of a single ten-input AND gate.

An interesting variation on the use of probabilistic approaches to testing is given by Savir, et al. [9,51]. The proposed technique is actually a design-for-testability tool rather than a test generation tool. The authors define the syndrome of the network to be the count of minterms in the function, thus equivalent to finding the probability that the output is 1 when input signal probabilities are 1/2. A syndrome-testable network is one that has a different syndrome from the good one in the presence of a fault. The authors show that a network implementing a unate function has such a property (first shown by Betancourt [13], and later shown by Akers [7] to be extendable, in a sense, to general logical functions). The innovation in Savir's technique is in the methods for determining whether a network is syndrome-testable, and in adding a minimum of extra control lines and gates in order to put it into syndrome-testable modes. Since his proposed test vector sets are the full 2^N possible combinations, it would be preferable to use the syndrome-testable design approach with a deterministic technique such as the "minimum test sets" given by Betancourt [13] for unate

networks, or even to extend the design approach to make the networks fanout-free, and then apply the BKA.

4.4 P-EXP APPROACH TO TESTING

4.4.1 SIGNAL VECTORS

P-exps can be manipulated to obtain the signal probabilities at any point in a network, either individually or jointly. We will generally assume a "bundling constraint" of 1/2 on each input, which is that achieved by uniform distribution of random test vectors.

The approach used here will be to apply deterministic methods, such as path sensitization, to describe a desired test, and then either to use P-exps to solve for the necessary input conditions (resulting in a test vector), or to find the expected testing length.

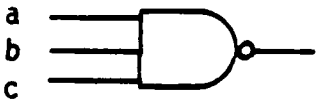
As stated earlier the deterministic methods generally assume that the signal at a node is faulty, and this signal (or its effect) is propagated to the output using one-, two-, or n-dimensional path sensitization. An alternative to this is to use a signal vector instead of a signal. A signal vector, in this respect, is an ensemble of signal values that are applied to a logical element. The notation that follows is that of Hayes [30]. In this case a test occurs when the desired signal vector is applied to the inputs of a gate, and the gate output signal is propagated to the output.

Test Recipes

For a single gate, Hayes used two types of tests, which he termed uniform and essential. For NAND gates, the uniform test, u, is a vector of all-ones applied to the inputs. For an n-input NAND gate, there are n essential tests, consisting of n-1 ones and a single zero, which we refer to as e_1, e_2,... e_n. Thus, there are n+1 tests required for a single gate. Figure 14 shows two-input and three-input NAND gates, with the respective u and e_i tests. We will, for convenience, use e_a, e_b, and e_c when these signal names have been assigned. We refer to each u and e_i test as being a separate signal vector. We will refer to the complete set of required tests for a gate as the testing recipe of the gate.

We will write signal vector values as bit strings. Thus, the signal vector 10 should be interpreted as the concatenation of "1" and "0," not as the numeric value, ten.

1	2	
1	1	- u
0	1	- e_1
1	0	- e_2



a	b	c	
1	1	1	- u
0	1	1	- e_a
1	0	1	- e_b
1	1	0	- e_c

Figure 14. Uniform and Essential Tests for NAND Gates.

AND gates have the identical testing recipe as NAND gates. For OR and NOR gates, the u test is the all-zero vector, and the e_i tests consist of a single one in a field of zeros. Two-input EXOR and EXNOR gates require all four signal vectors, so u and e_i tests are not directly applicable.

Fault Models

Although the u and e_i tests form a complete recipe for detecting all single stuck-at faults, they may not be sufficient in all cases. Galiay, Crouzet, and Vergniault [24] have given specific examples where MOS gate realizations are inadequately tested by this recipe. Further, most complex digital microcircuits are designed using standard cells that are optimized realizations of common functions (such as adders). These cells bear little resemblance to the gate-equivalent models. For these cells, one can devise recipes that check for 100% of the single stuck-at faults, but these are applied as signal vectors to the entire logical block, and they may have no resemblance to u and e_i tests. The P-exp approach that follows is not limited to u and e_i tests, but can be applied to any set of signal vectors.

In fairness to the classical stuck-at fault model, it should be noted that for microcircuits implemented in bipolar technologies the standard cells generally have a very close relationship to the gate-level representation. Experiments that involved the insertion of actual faults by laser beam [18] and comparison of the thus-modified output to that predicted by LASAR, a commercially-available digital network simulation program that uses the stuck-at fault model, have shown excellent correlation between the two. Case [16] has shown that for CMOS gates the stuck-at fault model accounts for all but 3.2% of the observable faults.

4.4.2 USING P-EXPS

Test generation using P-exps requires several steps. First, it is necessary to obtain the P-exp for every node in the network. Next, a particular gate is chosen, and the P-exp for one of the u or e_i tests is formed. The conditions for propagation to the output are listed, and their P-exps are formed. Finally, the P-exp describing the selected u or e_i test, and one or more P-exps for describing the conditions propagation are multiplied together, forming an overall P-exp that yields the probability that the test will occur.

Example 16: Consider the network in Figure 15, which is the same as that in Figure 1 with the additional annotation of the P-exps. We are interested in the u test for gate h. We know

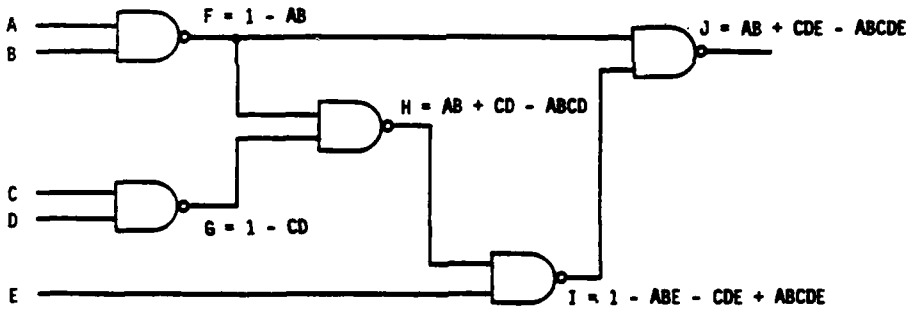


Figure 15. Digital Network for Example 16.

that the probability that both inputs to gate h will be 1 is given by the P-exp FG, which is also 1-H, or,

$$1 + ABCD - AB - CD$$

If we evaluate this expression with all probabilities equal to 1/2, we see that the u test is applied to gate h with probability 0.5625. However, the output of gate h must be propagated or sensitized to the output, and for this to occur then gates e and f must be 1. The P-exp for this event is

$$EF = E - ABE$$

which occurs with probability 0.375. The overall event, that of a u test being applied to gate h and being propagated to the output, is described by

$$\begin{aligned} & FGEF \\ & = EFG \\ & = ABCDE + E - ABE - CDE \end{aligned}$$

and the probability of this test occurring with equiprobable random inputs is 0.28125. This means that out of the 32 possible test vectors, 9 are valid u tests for gate h. The P-exp could also be solved for 1 at this point in order to find one of these tests.

On the other hand, let us try to apply the e_f test to gate h, which means that the condition fg = 01 is to be applied. The same propagation conditions are necessary, and we observe immediately that

$$\overline{FGEF} = 0$$

which means that we have shown that this particular test cannot occur.

(End of example)

The following two-part example will contrast the D-Algorithm with the P-exp approach:

Example 17: In Figure 16 we show a network which is Schneider's counterexample to one-dimensional path sensitization [53]. Applying the D-Algorithm to find a test for gate 6 SA0,

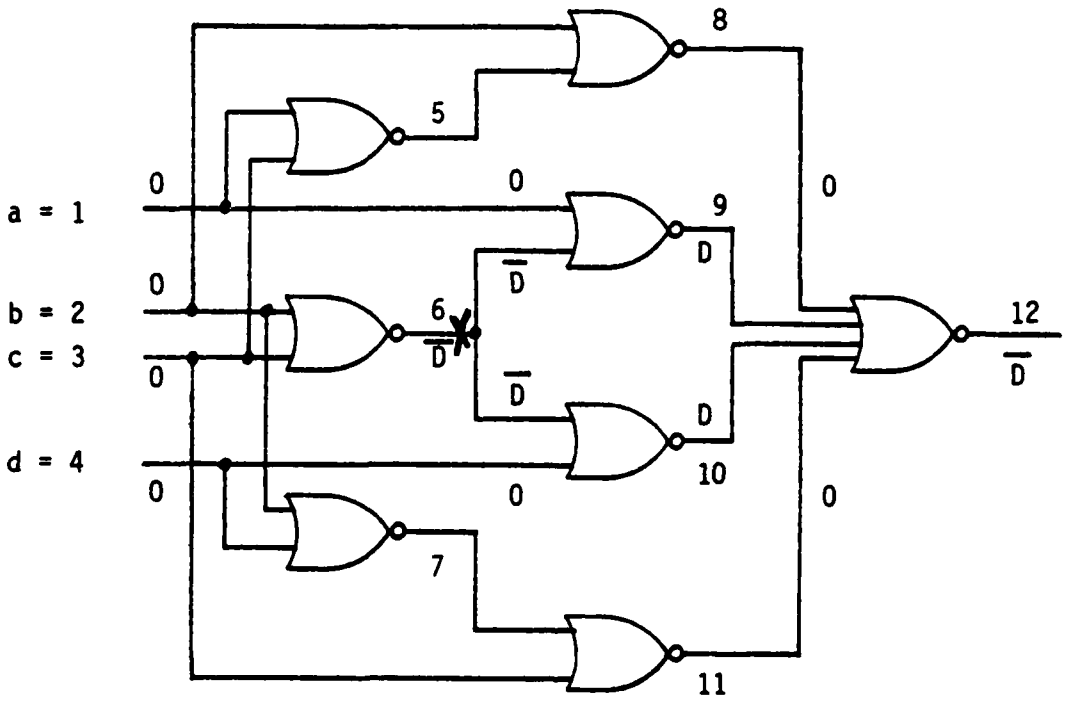


Figure 16. Schneider's Counterexample to One-Dimensional Path Sensitization.

we find that there is no single path to the output, and we are forced to try a two-dimensional path. We are able to propagate the $\overline{D}$ values through gates 9 and 10 simultaneously and then to the output, and the input vector 0000 provides the only such test.

For convenience in applying the P-exp approach, we have converted the NOR-network of Figure 16 into the NAND/AND-network of Figure 17, using mechanical application of DeMorgan's Theorem. This could have been reduced to a simpler all NAND-network, but we wish to retain a simple mapping between the signal lines in the original network and the new network. There are 30 signal lines named in the new network, where those numbered 1-12 correspond one-to-one with those in Figure 16. We will take a notational liberty and refer to the primary inputs 1, 2, 3, 4 with the additional names a, b, c, d. In order to generate a test for 6 SA0, we must make signal 6 a logical 1, and propagate the effect to the output. The P-exp for signal 6 is

$$6 = 1 + BC - B - C$$

For propagation we will consider only single (one-dimensional) path sensitization. Arbitrarily, we choose the upper branch, which is by way of gate 9 (actually it is through 22, 9, 28, and 12). For this to occur, we see that signal lines 21, 27, 29, and 30 must <u>all</u> be equal to 1. Note that it is simple for a human or computer program to derive this type of path tracing information (activation, propagation). It is difficult only to specify the input conditions which <u>cause</u> these desired internal conditions to occur (line justification). The P-exps for these signal lines are:

$$21 = 1 - A$$
$$27 = 1 + AB + AC + BC - ABC - A - C$$
$$29 = 1 + BC + BD + CD - BCD - B - C$$
$$30 = 1 + BC + BD + CD - BCD - B - D$$

When we AND or multiply these P-exps together, we find that the resulting P-exp (describing the test for 6 SA0), is:

$$1 + ABCD + AB + AC + AD + BC + BD + CD$$
$$- ABC - ABD - ACD - A - BCD - B - C - D$$

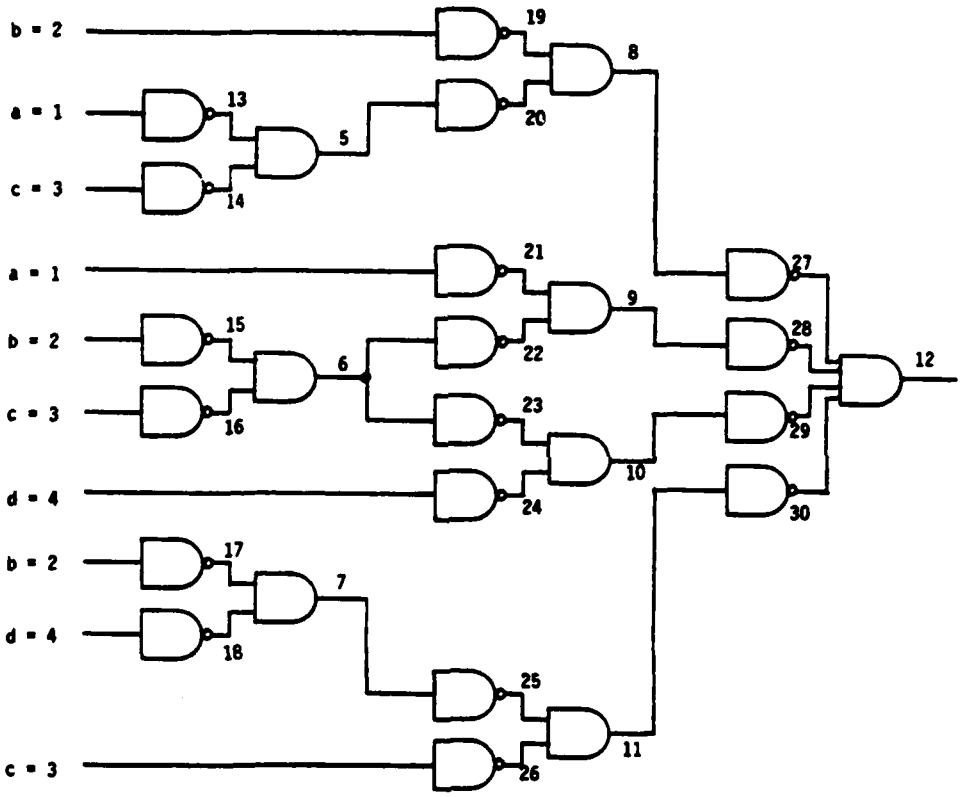


Figure 17. NAND/AND Realization of Network in Figure 16.

This expression, when evaluated with equiprobable inputs yields a probability of 0.0625, or 1/16. Therefore, there exists only one such test, and this probability expression can be solved to find that this test is 0000, just as was found previously by using the D-Algorithm.

(End of example)

In the D-Algorithm single paths are formed when a D or $\overline{D}$ signal is propagated by either 0 or 1 signals (0 for OR and NOR gates, and 1 for AND and NAND gates). Multiple paths are formed when D or $\overline{D}$ signals reconverge at a gate's inputs. From Figure 11 we see that two (or more) D signals will propagate each other, as will two (or more) $\overline{D}$ signals, through a NAND gate. Since in the presence of a fault both signals change their values, the output will also change. On the other hand, D and $\overline{D}$ inputs to the same gate will not be propagated.

Let us contrast this type of path sensitization with that done using P-exps. We select a fault site, and write P-exps that describe the joint event of causing a different value from the faulty one to appear at the fault site, and of the effect of this being propagated using 1 signals (for NAND and AND) or 0 (for NOR and OR). In Example 17 we did this successfully using what appeared to be one-dimensional sensitization, although we know that, from the D-Algorithm, this was actually two-dimensional. It happened that the faulty value took on the correct signal value to allow propagation. When this occurs, the path sensitization performed using P-exps will never need to be more than one-dimensional.

The P-exp approach may fail to find a test if the following occurs. A two-input NAND gate may have two $\overline{D}$ values at its inputs. From Figure 11 the D-Algorithm will correctly place a D at the output. However, using P-exps, we will be looking upon only one of these signals as being faulty, and since a $\overline{D}$ is actually a 0, the resulting P-exp will yield a zero probability of propagation through this gate.

4.4.3 EXPECTED TEST LENGTH

The P-exp approach provides a method for finding either a single test or all tests for a fault in a combinational network. The difficulty with techniques such as the D-Algorithm lay in the forward propagation and line justification processes. The difficulty with the approach given here is in the computational complexity of the P-exps. The P-exp technique would have its greatest utility in areas where probabilities are required.

One such area is the finding of expected test lengths, which is the average number of random test vectors required, on the average, before a test has been applied for a given fault. A related problem is to find the number of random test vectors required to reach an arbitrarily high probability, or testing confidence level, that the fault will have been detected (such as 99.9%). Another area is in answering probabilistic questions about problems that can be transformed into combinational networks or logical functions, such as the reliability of a communications network. Only the first two areas will be discussed here.

We will concern ourselves with random test generation that is performed both with and without replacement. Test Generation With Replacement (TGWR) models the real-world conditions that a logical network sees in operation. Test Generation Without Replacement (TGWOR) is the easier of the two testing methods to apply in practice, as the 2^N unique test vectors for an N-input network are readily obtained by using counters or maximal length pseudorandom shift registers. Analysis of TGWR will let us answer questions about fault latency in systems, or how long we may expect to wait before an existing fault causes a difference in the output. Analysis of TGWOR will aid us in answering questions about actual random testing of combinational networks.

We will subdivide the problem by talking about the case where we require several events to occur, for example, the fault detection experiment is complete when all three necessary u and e_i tests have been applied to a particular two-input gate. We will go even further, and deal separately with the case where the probability of the test we wish to occur is so small that only a single test vector out of 2^N provides it.

A taxonomy of our discussion is as follows:

- TGWR
  - single event
    - $p >> 2^{-N}$
    - $p = 2^{-N}$
  - compound event
    - probabilities nearly equal
    - one probability much smaller than the others

- TGWOR
  - single event
    - $p >> 2^{-N}$

- $p = 2^{-N}$

We will assume from this point that the numerical probabilities of the tests are given (obtained through the use of P-exps). We must emphasize that the probabilities include the conditions of both activation and propagation. Also, we will assume that the order in which the test vectors are applied does not modify the given probabilities. For example, suppose we have a vector of inputs to a network, (abcdefgh), where we must have a=1 in order to enable the outputs of the network. If we supply test vectors from a counter where a is the most significant bit we will have to wait until the one hundred twenty-ninth vector before anything of interest occurs. For this reason, a pseudorandom sequence generator is probably better than a simple counter.

4.4.3.1 TEST GENERATION WITH REPLACEMENT

Consider a set of test vectors numbered n = 0, 1, 2,... where vector 0 is the case that no test has been performed. With an N-input network, there are 2^N possible unique test vectors, but test vector generation is performed with replacement, so the possible number of test steps is infinite. We define success to be the event that a test has been applied for a given fault. Let p be the numeric probability yielded by the P-exp that describes the test condition for the given fault (thus, p is the probability of success on a single test), and let q = 1-p.

We define a random variable that is the number of the test on which success occurs. Since we wish to know the number of tests that must be applied before success is observed, the distribution is Geometric. The probability distribution function or cumulative distribution function will be denoted by P(n), which is the probability that success occurs on or before test vector n. The probability mass function, p(n), is P(n) -P(n-1), or the probability that success occurs for the first time on test vector n.

For a given value of p, the probability distribution function is given by [43]:

$$P(n) = 1-q^n \tag{7}$$

and the probability mass function is given by:

$$p(n) = pq^{n-1} \tag{8}$$

The expected test length, l, is

$$1 = \sum_{n=1}^{\infty} n\, p(n)$$

$$= \sum_{n=1}^{\infty} npq^{n-1}$$

$$= 1/p \qquad (9)$$

For this case we can use Eq. 9 to find the expected test length, or we can solve Eq. 7 to find the number of vectors required to reach an arbitrary level of testing confidence. In Example 17 we found that the test that detects signal 6 SA0 occurs with probability 0.0625, and Eq. 9 tells us that the expected test length is 1/0.0625 or 16.

Eq. 7 can be solved for n,

$$n = \log(1-P(n))/\log q \qquad (10)$$

where we can substitute for "P(n)" any desired testing confidence level.

For Example 17, we thus find that this test will occur with probability:

50% after 10.74 vectors
75% after 21.48 vectors,
99% after 71.36 vectors, and
99.9% after 107.03 vectors.

Shedletsky and McCluskey [55] discussed TGWR from the viewpoint that the most difficult fault to detect in a network is one that can be detected by only <u>one</u> of the 2^N possible test vectors (if it can be detected at all). In this case, $p = 2^{-N}$. They derived an expression that is the equivalent of Eq. 10, where $q = 1-2^{-N}$, and "P(n)" was the desired probability of observing the fault. They referred to the required number of test vectors as the <u>fault latency</u>.

Example 18: If N = 20 and we want to know the upper bound on the number of vectors, n, that are required to achieve a 90% probability of observing the least-observable fault, the expression is:

$$n = \log(0.1)/\log(1-2^{-20})$$

$$= 2.4144 \times 10^{6} \text{ vectors}$$

(End of example)

If we express this as a ratio with the possible number of unique vectors, 2^N, we find that the 90% point occurs at 2.3026 times the length of the exhaustive TGWOR pattern set.

It is interesting to observe the behavior of this ratio. If we denote this ratio by R, and by C the desired probability of observing the fault (confidence level), then we have from Eq. 10,

$$R = \ln(1-C) \bullet \frac{2^{-N}}{\ln(1-2^{-N})}$$

(Here it is desirable to use natural logarithms explicitly.) We can apply l'Hospital's Rule and find that, in the limit, for large N (N > 10),

$$R = -\ln(1-C)$$

which tells us that the ratio is

R = 2.3026, for C = 90%,
R = 2.9957, for C = 95%,
R = 4.6052, for C = 99%, and
R = 6.9078, for C = 99.9%.

Simple vs. Compound Events

The previous problems may be categorized as waiting for a simple event, or a single event, to occur. Many simultaneous conditions had to be met, as shown in the manipulation of the P-exps leading to the P-exp describing the SA0 test, but that test is a single event. On the other hand, complete testing of a gate or a functional block in a network generally

requires a set of signal vectors (the testing recipe) to be applied. A compound event, then, is the condition in which the complete required set of signal vectors, in any order, has been applied to a given part of the network. We will include in the probabilities, as always, the necessary condition that the effect of the test has been propagated to the output.

An exact solution for the compound event problem may be found by using Markov Matrices [31]. Consider the case where there are four mutually-exclusive events that may occur, {a, b, c, d}, and we will define success as being the condition when the three events a, b, and c have each occurred at least once. An example of this is the testing of a two-input gate, where there are four possible signal vectors, and a complete test has occurred when three of them (the u, e_1, and e_2 tests) have been applied. Since the network is combinational there is no memory involved, and the order of test application is immaterial.

For this set of four events, let us define a set of corresponding numeric probabilities: {A, B, C, D}, where A = P(a), B = P(b), C = P(c), and we define D = 1-A-B-C. (Mnemonically, we can consider "d" to be the "don't care," "default," or "doesn't help" case) From the set of the three events that interest us, we can form the power set (the set of all distinct subsets) which consists of 2^3 events:

{ $\emptyset$, a, b, c, ab, ac, bc, abc}

where "$\emptyset$" represents the null event, denoting the condition where we have observed either nothing or only d's. The event "ab," for example, represents the case where we have observed one or more a's and b's, none or more d's, and no c's. We can now form an 8x8 Markov Matrix, treating the events as states, because we know from the individual probabilities what any transition probability will be. If we are in state ab the probability of going to abc on the next transition is C, while the probability of remaining in ab is A+B+D. Note again that the upper-case characters here represent numeric probabilities, not P-exps, although the numeric values will in general be derived from the appropriate P-exps. The complete Markov Matrix for the general three-compound event case is shown in Figure 18. Let us refer to this matrix as P. The row headings correspond to "starting states," and the column headings to "next states." The matrix P, or P^1, tells us, for example, that given that we started in state ab, the probability that we will reach abc after one transition is C. P is thus the one-step transition matrix. The multistep transition probabilities are found by raising P to higher powers, where P^2, P^3, P^4, and so on, are obtained by repeated matrix multiplication of P with itself. In particular, we are interested only in the case where we start in the null state, so we need

Starting State \ Next State	∅	a	b	c	ab	ac	bc	abc
∅	D	A	B	C	0	0	0	0
a	0	A+D	0	0	B	C	0	0
b	0	0	B+D	0	A	0	C	0
c	0	0	0	C+D	0	A	B	0
ab	0	0	0	0	A+B+D	0	0	C
ac	0	0	0	0	0	A+C+D	0	B
bc	0	0	0	0	0	0	B+C+D	A
abc	0	0	0	0	0	0	0	1

Figure 18. Markov Matrix, P, for Three-Compound Event Case.

multiply only the first row by the P matrix. If we refer to the first row of P as the 1x8 vector V, or V(1), then the multistep transition probabilities are simply

$$V(2) = V(1)P$$
$$V(3) = V(2)P$$
$$\vdots$$
$$V(n) = V(n-1)P$$

where the right-hand side represents the cross product of a 1x8 vector with an 8x8 matrix. The eighth or last element of V(n) will be the n^{th} value of the probability distribution function that gives the probability that all three of the events have occurred on or before step n, given that the starting condition was the null testing condition.

Although a closed form expression for the multistep transition probabilities can be obtained using z-transforms [31], it was simpler for the purpose of this work to calculate the various V(i) vectors directly in order to gather statistics about test lengths.

It was found that, in the case where A = B = C, the expected test length was approximately

$$l_{abc} = 1.833/A \qquad (11)$$

as opposed to 1/A that would result from using Eq. 9 in the simple event case. The constant of proportionality was different when the three event probabilities differed from each other.

Example 19: Using the Markov Matrix method, when A = B = C = 0.01, we obtain l_{abc} = 183.2738, and the three events would be observed with probabilities:

50% at n = 158 (0.8621 * l_{abc}),
99% at n = 568 (3.0992 * l_{abc}),
99.9% at n = 797 (4.3487 * l_{abc}),

and when A = B = C = 1/3, we obtain l_{abc} = 5.4989, and the testing confidence levels were:

50% at n = 5 (0.9093 * l_{abc}),

99% at n = 15 (2.7278 * l_{abc}), and
99.9% at n = 20 (3.6371 * l_{abc}).

(End of example)

The expected test length, then, provides a reasonable means of estimation of these other confidence levels.

Approximation Method for the Compound Event Case

The Markov Matrix method is useful because it provides an exact answer, but lengthy computations are required in the absence of a closed form expression. In Example 19, in order to reach four decimal places of precision when A = B = C = 0.01 the computer program had to perform 1095 iterations (although for A = B = C = 1/3 it performed only 28). We will often be interested in probabilities on the order of 2^{-20} or 2^{-50} (10^{-6} or 10^{-15}). It would be desirable then to use an approximation that has an easily-obtainable closed form expression, and, more importantly, could be easily extended to more than three events.

We wish to calculate

$P_{abc}(n)$ = Prob{a, b, and c have all occurred on or before step n}

Let us use the notation $p_a = P(a)$ and let $q_a = 1-p_a$. We will approximate the exact probability distribution function for the compound event by taking the product of the individual probability distribution functions. Since the events a, b, and c are mutually-exclusive, this operation is not strictly correct. However, if the probabilities of these events are very small, then we may justify this approximation (that the events are not mutually-exclusive) on the basis that the joint probabilities become vanishingly small. This is shown later to be a good approximation.

For the three-event case, we have:

$$P_{abc}(n) = P_a(n)\, P_b(n)\, P_c(n)$$

$$\cong (1-q_a^{\,n})(1-q_b^{\,n})(1-q_c^{\,n})$$

$$= 1 - q_a^{\,n} - q_b^{\,n} - q_c^{\,n}$$

$$+ (q_a q_b)^n$$

$$+ (q_a q_c)^n$$

$$+ (q_b q_c)^n$$

$$- (q_a q_b q_c)^n \qquad (12)$$

We can then obtain the probability mass function by subtracting $P_{abc}(n-1)$ from this sum:

$$p_{abc}(n)$$

$$\cong q_a^{n-1}(1-q_a)$$

$$+ q_b^{n-1}(1-q_b)$$

$$+ q_c^{n-1}(1-q_c)$$

$$- (q_a q_b)^{n-1}(1-q_a q_b)$$

$$- (q_a q_c)^{n-1}(1-q_a q_c)$$

$$- (q_b q_c)^{n-1}(1-q_b q_c)$$

$$+ (q_a q_b q_c)^{n-1}(1-q_a q_b q_c) \qquad (13)$$

The expected test length is

$$l_{abc} = \sum_{n=1}^{\infty} n\, p_{abc}(n)$$

and applying the identity

$$\sum_{n=1}^{\infty} n\, y^{n-1}(1-y) = \frac{1}{1-y}$$

to each term of the approximation for $p_{abc}(n)$, we have

$$
\begin{aligned}
& I_{abc} \\
& \cong 1/(1-q_a) \\
& + 1/(1-q_b) \\
& + 1/(1-q_c) \\
& - 1/(1-q_a q_b) \\
& - 1/(1-q_a q_c) \\
& - 1/(1-q_b q_c) \\
& + 1/(1-q_a q_b q_c)
\end{aligned} \tag{14}
$$

Note that Eq. 14 is similar in form to Eq. 3, the General Additive Law of Probability. Extension of this result to problems with more than three events (or to two events) is straightforward.

Eq. 12 is difficult to solve for n, but for $q_a = q_b = q_c = q$, we have

$$n = \frac{\log(1 - \sqrt[3]{P_{abc}(n)})}{\log q} \tag{15}$$

which is similar in form to Eq. 10, which was for the simple event case.

Example 20: We can contrast the performance of the exact and approximate methods by using the probabilities used in Example 19. When Eq. 14 is evaluated with A = B = C = 0.01 ($q_a = q_b = q_c = 0.99$), we obtain $I_{abc} = 182.9151$, which differs from the exact value obtained earlier by only 0.1957%. The worst case appears when A = B = C = 1/3, resulting in $I_{abc} = 5.0211$, which underestimates the true I_{abc} by 8.69%.

Applying Eq. 14 when q = 0.99, and finding the number of test vectors required to achieve a testing confidence level of 99.9%, we obtain n = 796.5937, which underestimates the exact n by 0.05% (but the exact figure of 797 was the first integer value that exceeded 99.9%). In the worst case, q = 2/3, we calculate n = 19.7453, which compares favorably to the

exact value of 20. These results are so close that if we take the ceiling of the calculated values of n we obtain the correct integer values.

(End of example)

Example 21: We can attempt to apply the compound event technique to the network in Figure 16, and to its transformed version in Figure 17. As shown earlier, NOR gate 6 can be tested for SA0 with probability 0.0625. The output having the value 1 corresponds to the input signal vector 00. The other two necessary signal vectors are 01 and 10. When we evaluate the P-exps for them, they both occur with probability 0.0. Therefore, there will be a number of undetectable faults associated with this gate, and we may rate this gate overall as being untestable.

(End of example)

Example 22: Let us select another gate and try this technique. NOR gate 9 presents a challenge, as it is separated both from the primary inputs and outputs. Referring to Figure 17, the only possible path for sensitization to the output is by way of gates 28 and 12, which will require ones at the outputs of gates 27, 29, and 30. The P-exp describing this propagation condition is

$$1 + ABCD + AB + AC + AD + BC + BD + CD$$
$$- ABC - ABD - ACD - A - B - C - D$$

The signal vector 00 occurs when gates 21 and 22 have ones as their outputs. ANDing these conditions with that for propagation, above, the P-exp for this test is

$$BCD - ABCD$$

which occurs with probability 1/16, and the only such test vector is abcd = 0111.

For signal vector 01, we need 21 and 6 to be ones, and the conditions for propagation must be satisfied. The P-exp is

$$1 + ABCD + AB + AC + AD + BC + BD + CD$$
$$- ABC - ABD - ACD - A - BCD - B - C - D$$

which evaluates to 1/16 (cumulative test probability is 2/16), and that test vector is 0000.

For signal vector 10, input 1 and gate 22 must be ones, and with the conditions for propagation the P-exp is

ABCD

which has probability 1/16 (cumulative test probability is 3/16), and the test vector is 1111.

(End of example)

Note that, in Example 22, we paid attention to the original probability of propagation to the output, and then to the cumulative probabilities of the tests as we generated them. Since the signal vectors are mutually exclusive, the sum of their probabilities cannot exceed the probability of propagation. The propagation probability above was 3/16, which tells us that no more than three such test vectors can exist. The three vectors we found had probabilities totaling 3/16. If the first one, for example, had had a probability of 2/16, we would deduce immediately that tests providing all three required signal vectors could not exist, and so the gate would be untestable, overall.

Example 23: From Example 22 we have found the probabilities of our three signal vectors of interest. We can apply Eq. 14, letting

$$q_a = q_b = q_c = 15/16$$

which results in an average test length of $l_{abc} = 28.9068$.

We may also apply Eq. 13 to find the number of test vectors required to reach a given confidence level, since the probabilities are identical. To reach

50% requires 24.46 vectors,
99% requires 88.33 vectors, and
99.9% requires 124.05 vectors.

(End of example)

Compound Event With $p_a << p_b, p_c$

The Markov Matrix method and the approximation method were demonstrated above using probabilities that were all equal. While the results hold for the case where, for example, $p_a << p_b$ and $p_a << p_c$ (except of course for Eq. 14), it is interesting to note that when one of the probabilities is much smaller than the others this problem simplifies to the single event problem, where the smallest probability will dominate.

We can demonstrate this using the Markov Matrix method:

Example 24: In the case where A = B = C = 0.1, use of the Markov Matrix results in l_{abc} = 18.328, which is in accordance with Eq. 11. If we change A to 0.01, keeping B and C constant, then the new l_{abc} is 101.526. Changing A to 0.005 brings l_{abc} to 200.727.

(End of example)

We see that making A small in relation to B and C causes l_{abc} to behave more in accordance with Eq. 9, which governed the single event case.

4.4.3.2 TEST GENERATION WITHOUT REPLACEMENT

We will deal only with the single event case in this section. As mentioned earlier, TGWOR is of greater interest as a testing approach than is TGWR.

Let p be the initial probability that is given to us. Let p_i be the probability of detection on test vector i, given that the fault was not detected previously, and $p = p_1$. We then define:

$q = 1-p$

$q_i = 1-p_i$

$M = 2^N$ = number of unique test vectors.

$r = pM$ = number of vectors that detect the fault.

$b = qM$ = number of vectors that do not detect the fault.

From these definitions, we have:

$p_1 = r/M$
$q_1 = b/M$

$p_2 = r/(M-1)$
$q_2 = (b-1)/(M-1)$

$p_3 = r/(M-2)$
$q_3 = (b-2)/(M-2)$

.
.
.

$p_i = r/(M-i+1)$
$q_i = (b-i+1)/(M-i+1)$

$i = 1, 2, \ldots (M-r+1)$

The probability distribution function is then

$P(n) = 1-q_1q_2q_3\cdots q_n$

$n = 1, 2, 3, \ldots (M-r+1)$

$$= 1 - \frac{b(b-1)(b-2) \ldots (b-n+1)}{M(M-1)(M-2) \ldots (M-n+1)}$$

$$= 1 - \frac{b!\ (M-n)!}{M!\ (b-n)!} \qquad (16)$$

As a check on our assumptions we applied the Hypergeometric distribution [43] and obtained the identical result (r = red, b = black).

Eq. 16 is not easily solved for n. As a result, we wrote a computer program that implemented this equation with a binary search to yield the value of n for the parameters N and p. Since we were interested in values of N that were large, the factorials were

cumbersome. In order to avoid overflow we used the logarithms of the factorials. We computed exact factorials only for arguments less than 100. Otherwise, we used Stirling's Formula,

$$n! \cong \sqrt{2*\pi*n} * n^n * e^{-n}$$

which is quite accurate for large arguments.

We contrasted the values of n obtained in this manner with those obtained by TGWR (Eqs. 7 and 9). Testing always had a higher confidence level using TGWOR from the second test vector on, because the probability of detection increases monotonically as testing progresses. Also, TGWOR results in a worst-case test length of M-r+1, whereas TGWR may never detect the fault in our lifetimes.

We found in TGWOR that for large N (N > 10), and for p relatively large in relation to 2^{-N}, the confidence levels can be calculated quite accurately using Eq. 7, and the expected test length by Eq. 9, thus reducing the problem to the TGWR case. On the other hand, when p was small, taking the worst-case value of $p = 2^{-N}$, then for a given confidence level C the values for n were accurately given by

$$n = C\ 2^{-N} \tag{17}$$

which says that if one needs a confidence level of 90%, then 90% of the total number of unique vectors must be applied.

We can contrast the number of tests required to reach a given confidence level using TGWR versus those required by the TGWOR method. Let us define G to be the gain in test vectors required by TGWR over TGWOR, for large N. For relatively large p, G is nearly unity. For $p = 2^{-N}$, we take the ratio between Eq. 7 and Eq. 17,

$$G = \frac{\ln(1-C)}{C} \cdot \frac{2^{-N}}{\ln(1-2^{-N})}$$

and by using l'Hospital's Rule arrive at

$$G = -\ln(1-C)/C$$

We have tabulated this gain factor for several values of C:

C = 25%, G = 1.1507
C = 50%, G = 1.3863
C = 75%, G = 1.8484
C = 90%, G = 2.5584
C = 95%, G = 3.1534
C = 99%, G = 4.6517
C = 99.9%, G = 6.9147
C = 99.99%, G = 9.2113

We see that, even though G would approach infinity for C approaching unity, the gain is still reasonable for reasonable confidence levels. Thus, while both TGWR and TGWOR are expensive, TGWR is not too much more expensive.

V. SUMMARY

In this report we have presented Probability Expressions as an alternate form for a logical function. We have done this in a manner similar to the time and frequency domain analysis of signals. We have presented theorems covering both the properties and the transforms of the P-exps.

We have discussed the complexity considerations of P-exp manipulations and have shown how these can be implemented efficiently on a computer. We have shown how P-exps can be used to aid in test generation for digital networks, and have presented some general results dealing with random testing.

The P-exp approach to digital network testing is probably not a practical means of stuck-fault test generation for large networks. However, new design styles are resulting in small logical blocks that may be treated separately, bringing the problem to a tractable level. The ease of implementing the P-exp approach over the D-Algorithm, or even over a Boolean expression approach, may outweigh some performance objections.

Many problems in other areas can be reduced to a problem that is equivalent to that of calculating the signal probability in a digital network. An example of this is a communications network where nodes and/or links have associated with them known failure probabilities, and one desires to find the probability that at least one complete path exists between a given pair of nodes. This problem has been addressed extensively in the literature. Fratta and Montanari [22] used Boolean algebra to aid in calculation of the probability, and Schneeweiss [52] actually used P-exps to directly compute the probability. The work described in this report can make it possible to do these calculations in a more efficient manner, and it can also be used in answering more complicated questions than simply the overall probability of failure.

APPENDIX

A number of small computer programs were written to aid in the work described in this report, and in related work. We will list some of the more important of these programs, and briefly describe their functions. All of these programs are written in PL/I and run under the Honeywell Multics Operating System.

PEXP: This program accepts as its input a description of a combinational network in terms of NAND and AND gates, and generates the P-exp for the output of each gate. The user may evaluate a P-exp with any desired set of input signal probabilities, and can solve an expression to find sets of inputs that yield a 0 or 1 output. This program prints the CPU seconds used in the generation of each P-exp, and also prints statistics about the maximum and final lengths of expressions.

The input to the PEXP program is contained in an ASCII text file. The first line of the file always consists of a list of the independent variable names that will be used. The rest of the file containes commands that either generate new P-exps or manipulate existing P-exps.

The commands currently implemented are as follows:

Pz & Pa Pb Pc ...

This will AND the P-exps Pa, Pb, Pc, ... and assign the result to the name "Pz." The P-exp is displayed, along with some statistics about it such as length and CPU time required for generation.

Pz ^ Pa Pb Pc ...

NAND.

P = Va Vb Vc ...

Evaluate the P-exp, P, with the first independent variable probability set to the numeric value Va, the second set to Vb, etc. The last value is replicated as many times as necessary in order to match the number of independent input variables. Thus, "P = 0.5" assigns all input probabilities to 0.5.

P ==

Evaluate the P-exp, P, with the input probabilities that were most recently set. Initially, the values are set to 0.5.

P - 1

Find all input combinations for which the P-exp, P, has the output 1.0. This does not use the algorithm described in Section 3.5, but rather applies all input combinations, resulting in long run times, and voluminous output.

P - 0

Solve for 0.0.

*

When "*" is encountered in the input file, PEXP goes into a keyboard entry mode, where the user can interact with the program. When the user types "*" PEXP reverts to file input.

/ text

Comment.

The program terminates when it encounters the end of the input data file.

LASAR-EXPANDER: This program accepts as its input a description of a digital network in the LASAR [18] simulation language format, containing submodels, sub-submodels, etc., and outputs the same model in a "squeezed-out" form. This transformed form of the network model has only a main model composed of a single level of submodels. All sub-submodels (and below) are expanded into their full form (the gate level) in the submodels. If a completely expanded model is required then the user merely places the main model in a submodel (which is done by adding only six lines of text). This program is used to help convert network models existing at RADC into the input form for the PEXP program.

BKA: This program performs many functions. Its input is a LASAR model in the squeezed-out form provided by the LASAR-EXPANDER. The network elements are sorted into an order suitable for sequential evaluation, and it is determined whether the network is fanout-free.

If the network is fanout-free, this program can perform the Berger-Kohavi Algorithm [12], producing both the test counts and the tests themselves. For any strictly combinational network (one containing no feedback) the BKA program can output the network in the form required by the PEXP program.

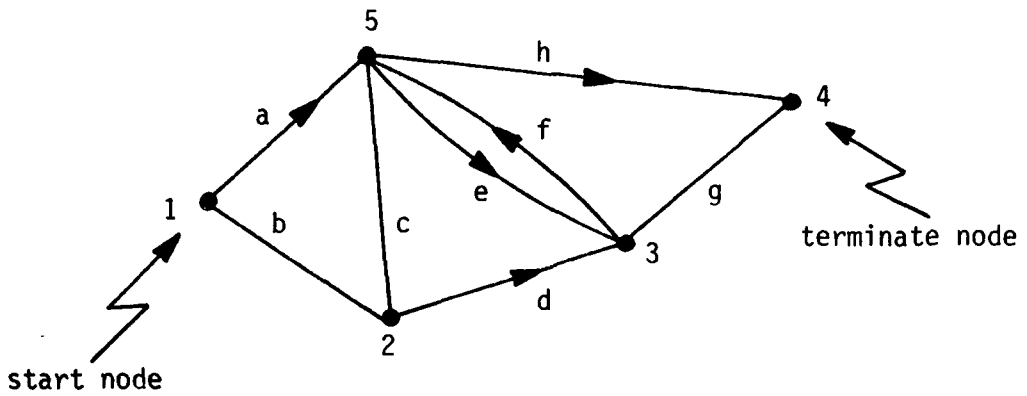
GENERATE: This program produces the logical functions used in Section 3.3 for calculating the average lengths of P-exps. This program generates all functions of two and three variables, and 500 randomly-selected functions of four through seven variables. The user can generate either the general function itself, or can ask for the monotonically-increasing unate form of the function. These unate functions are produced by the minterm covering relationship defined by McNaughton [42].

MARKOV-MATRIX: This program uses the Markov Matrix P (defined in Section 4.4.3.1, Figure 18) for the general three-compound event case. The user enters the probabilities of the events. The program computes the probability of success on any step n, finds the number of steps required to reach several different confidence levels (50%, 99%, and 99.9%), and computing the expected test length, l_{abc}. This program compares the exact answers, obtained using the Markov Matrix, to the approximations used in Eqs. 12 and 14, showing the percentage errors.

WOREPL: This program computes statistics about Test Generation Without Replacement. This program uses Eq. 16, defined in Section 4.4.3.2.

ALLPATHS: This program finds all simple paths in a model of a communications network. The output of this program is in the format required for input to PEXP.

The graph in Figure 19 shows nodes connected by directed and undirected arcs, the input required by ALLPATHS, and the output of ALLPATHS. ALLPATHS produces only paths, not cutsets. The algorithm used is simple and is potentially of O(N!) complexity, where N is the number of nodes. This complexity depends on the degree of connectivity of the graph.



Input:

```
1 a 5, b 2
2 b 1, c 5, d 3
3 f 5, g 4
4 g 3
5 c 2, e 3, h 4
s = 1
t = 4
```

Output:

```
/ Filename "fig19a"
/ from "1" to "4"

 a b c d e f g h

   1 * a c d g
   2 * a e g
   3 * a h
   4 * b c e g
   5 * b c h
   6 * b d f h
   7 * b d g

8 ^ 1 2 3 4 5 6 7

/ 5 nodes
/ 8 arcs
/ 7 paths
/ 25 visits
/ shortest path length = 2
/ longest path = 4
/ average path =     3.28571
```

Figure 19. ALLPATHS Input and Output.

REFERENCES

The following abbreviations for the names of periodicals and journals have been used:

BSTJ Bell System Technical Journal

IBMJ IBM Journal of Research and Development

IEEETC IRE or IEEE Transactions on Electronic Computers (before 1968), IEEE Transactions on Computers (1968 to present)

IEEETR IEEE Transactions on Reliability

FTC, FTCS Fault Tolerant Computing Symposium (Note that the last digit represents the year, e.g., FTCS-9 and FTCS-11 were published in 1979 and 1981, respectively)

1. P. Agrawal and V. D. Agrawal, "On improving the efficiency of Monte Carlo test generation," FTC-5, pp. 205-209.

2. P. Agrawal and V. D. Agrawal, "Probabilistic analysis of random test generation method for irredundant combinational logic networks," IEEETC, pp. 691-695, July 1975.

3. P. Agrawal and V. D. Agrawal, "On Monte Carlo testing of logic tree networks," IEEETC, pp. 664-667, June 1976.

4. V. D. Agrawal, "When to use random testing," IEEETC, pp. 1054-1055, November 1978.

5. V. D. Agrawal, "An information theoretic approach to digital fault testing," IEEETC, pp. 582-587, August 1981.

6. A. N. Airapetian and J. F. McDonald, "Improved test set generation algorithm for combinational circuit control," FTCS-9, pp. 133-136.

7. S. B. Akers, Jr., "Universal test sets for logic networks," IEEETC, pp. 835-839, September 1973.

8. D. B. Armstrong, "A deductive method for simulating faults in logic circuits," IEEETC, pp. 464-471, May 1972.

9. Z. Barzilai, J. Savir, G. Markowsky, and M. Smith, "VLSI self-testing based on syndrome techniques," 1981 IEEE International Test Conference (Cherry Hill '81), pp. 102-109.

10. R. G. Bennetts, "On the analysis of fault trees," IEEETR, pp. 175-185, August 1975.

11. R. G. Bennetts, "Referee's comments to 'Calculating the probability of Boolean expression being 1'," IEEETR, pp. 19-21, April 1977.

12. I. Berger and Z. Kohavi, "Fault detection in fanout-free combinational networks," IEEETC, pp. 908-914, October 1973.

13. R. Betancourt, "Derivation of minimum test sets for unate logical circuits," IEEETC, pp. 1264-1269, November 1971.

14. G. Boole, An Investigation of the Laws of Thought, on which are Founded the Mathematical Theories of Logic and Probabilities, 1854.

15. M. A. Breuer and A. D. Friedman, "Functional level primitives in test generation," IEEETC, pp. 223-235, March 1980.

16. G. R. Case, "Analysis of actual fault mechanisms in CMOS logic gates," 13th IEEE, ACM Design Automation Conference, pp. 265-270, 1976.

17. C. W. Cha, W. E. Donath, and F. Özgüner, "9-V algorithm for test pattern generation of combinational digital circuits," IEEETC, pp. 193-200, March 1978.

18. W. H. Debany, Jr., et al., Test Generation and Fault Isolation for Microprocessors and Their Support Devices, RADC-TR-80-274 (NTIS #AO 96360), November 1980.

19. W. Dobosiewicz, "Sorting by distributive partitioning," Information Processing Letters, Vol. 7, No. 1, pp. 1-6, 12 January 1978.

20. C. R. Edwards, "The application of the Rademacher-Walsh transform to Boolean function classification and threshold logic synthesis," IEEETC, pp. 48-62, January 1975.

21. C. R. Edwards and S. L. Hurst, "A digital synthesis procedure under function symmetries and mapping methods," IEEETC, pp. 985-997, November 1978.

22. L. Fratta and U. G. Montanari, "Boolean algebra method for computing the terminal reliability in a communication network," IEEETC, pp. 203-211, May 1973.

23. H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," IEEETC, pp. 555-560, June 1982.

24. J. Galiay, Y. Crouzet, and M. Vergniault, "Physical versus logical fault models MOS LSI circuits: Impact on their testability," IEEETC, pp. 527-531, June 1980.

25. M. R. Garey and D. S. Johnson, Computers and Intractability (A Guide To The Theory Of NP-Completeness), San Francisco, CA: W. H. Freeman and Co., 1979.

26. D. D. Givone, Introduction to Switching Circuit Theory, NY: McGraw-Hill, 1970.

27. P. Goel, "Automatic test generation for VLSI: Techniques. results, and projections." 1982 IEEE International Automatic Testing Conference (AUTOTESTCON '82), pp. 260-269.

28. T. Hailperin, Boole's Logic and Probability. Studies in Logic and Foundations of Mathematics; Vol. 85, NY: North-Holland Publishing Co., 1976.

29. C. R. P. Hartmann, P. K. Varshney, and W. H. Debany, Manuscript in preparation.

30. J. P. Hayes, "A nand model for fault diagnosis in combinational networks," IEEETC, pp. 1496-1506, December 1971.

31. R. A. Howard, Dynamic Probabilistic Systems, Vol I: Markov Models, NY: John Wiley & Sons, Inc., 1971.

32. R. B. Hurley, "Probability maps," IEEETR, pp. 39-44, September 1963.

33. O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," IEEETC, pp. 242-249, March 1975.

34. P. J. Janus, Adaptive Methods For Unknown Distributions In Distributive Partitioning Sorting, Master's Thesis, University of Rhode Island, published as "Algorithmic Complexity, Part 5," in Algorithmic Complexity, Vol. II, RADC-TR-82-152, June 1982.

35. G. J. Klir, Introduction To The Methodology Of Switching Circuits, NY: D. van Nostrand Co., 1972.

36. S. K. Kumar and M. A. Breuer, "Probabilistic aspects of Boolean switching functions via a new transform," Journal of the ACM, pp. 502-520, July 1981.

37. Z. Kohavi, Switching and Finite Automata Theory, NY: McGraw-Hill, 1978.

38. C. Y. Lee, "Analysis of switching networks," BSTJ, pp. 1287-1315, November 1955.

39. S. C. Lee, Modern Switching Theory and Digital Design, Englewood Cliffs, NJ: Prentice-Hall, 1978.

40. Y. H. Levendel and P. R. Menon, "Test generation algorithms for nonprocedural computer hardware description languages," FTCS-11, pp. 200-205.

41. Y. H. Levendel and P. R. Menon, "Test generation algorithms for computer hardware description languages," IEEETC, pp. 577-588, July 1982.

42. R. McNaughton, "Unate truth functions." IEEETC, pp. 1-6, March 1961.

43. W. Mendenhall and R. L. Scheaffer, Mathematical Statistics With Applications, North Scituate, MA: Duxbury Press, 1973.

44. A. Nijenhuis and H. S. Wilf, Combinatorial Algorithms, NY: Academic Press, 1975.

45. A. Papoulis, Signal Analysis, NY: McGraw-Hill, 1977.

46. K. P. Parker and E. J. McCluskey, "Analysis of logic circuits with faults using input signal probabilities," FTC-4, pp. 1-8 -- 1-12.

47. K. P. Parker and E. J. McCluskey, "Analysis of logic circuits with faults using input signal probabilities," IEEETC, pp. 573-578, May 1975.

48. K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," IEEETC, pp. 668-670, June 1975.

49. S. T. Ribeiro, "Random-pulse machines," IEEETC, pp. 261-276, June 1967.

50. J. P. Roth, "Diagnosis of automata failures: A calculus and a method," IBMJ, pp. 278-291, July 1966.

51. J. Savir, "Syndrome-testable design of combinational circuits," IEEETC, pp. 442-451, June 1980. "Correction to 'Syndrome-testable design of combinational circuits'," IEEETC, pp. 1012-1013, November 1980.

52. W. G. Schneeweiss, "Calculating the probability of Boolean expression being 1," IEEETR, pp. 16-19, April 1977.

53. P. R. Schneider, "On the necessity to examine d-chains in diagnostic test generation," IBMJ, p. 114, January 1967.

54. H. D. Schnurman, E. Lindbloom, and R. G. Carpenter, "The weighted random test-pattern generator," IEEETC, pp. 695-700, July 1975.

55. J. J. Shedletsky and E. J. McCluskey, "The error latency of a fault in a combinational digital circuit," FTC-5, pp. 210-214.